OLVASÓTERMI PÉLDÁNY TK 155. 368

KFKI-1982-24

GLADKIH IRINA
ZIMÁNYI MAGDOLNA

FORMAC

PROGRAMOZÁSI NYELV MATEMATIKAI KIFEJEZÉSEK SZIMBOLIKUS KEZELÉSÉRE

Hungarian Academy of Sciences

CENTRAL
RESEARCH
INSTITUTE FOR
PHYSICS

BUDAPEST

2017

KFKI-1982-24

F O R M A C

PROGRAMOZÁSI NYELV

MATEMATIKAI KIFEJEZÉSEK SZIMBOLIKUS KEZELÉSÉRE

Gladkih Irina, Zimányi Magdolna

Központi Fizikai Kutató Intézet
1525 Budapest 114, Pf.49

HU ISSN 0368 5330
ISBN 963 371 910 0

ABSTRACT

FORMAC is a high-level programming language for symbolic computation with mathematical expressions.

The capabilities of the language include computations with system-defined and user-defined functions, symbolic differentiation, substitution, analysis and comparison of expressions, automatic and user-controlled simplification.

FORMAC is an extension of the PL/I language.

This report serves as an user's manual to the FORMAC language and its use under the IBM System/360 OS operating system.

АННОТАЦИЯ

ФОРМАК - это язык программирования высокого уровня, созданный на основе ПЛ/I, и дающий возможность преобразования символьных математических выражений.

В языке ФОРМАК могут быть использованы функции, определенные системой или по указанию пользователя, выражения могут быть аналитически продифференцированы, вычислены, заменены другими выражениями, сравнены друг с другом, разделены на части и упрощены автоматически или по желанию пользователя.

Данная работа является инструкцией для изучения языка ФОРМАК и для его использования в системе ОС IBM/360.

KIVONAT

A FORMAC nyelv egy magasszintü programozási nyelv matematikai kifejezésekkel való szimbolikus számitások végzésére.

A FORMAC nyelvben használhatunk a rendszer által és a felhasználó által definiált függvényeket, kifejezéseket szimbolikusan differenciálhatunk, kifejezéseket részeikre bonthatunk, két kifejezést egymással összehasonlithatunk, és lehetséges kifejezéseknek automatikus vagy a felhasználó által irányitott egyszerüsitése.

A FORMAC nyelv a PL/I nyelv egy kiterjesztése.

A jelen report felhasználói kézikönyvként szolgál a FORMAC nyelvhez és használatához az IBM System/360 OS operációs rendszerben.

Tartalomjegyzék

Bevezetés

A FORMAC /FORmula MAnipulation Compiler/ magasszintü formulamanipulációs programozási nyelv. Segitségével matematikai kifejezéseket szimbolikusan kezelhetünk, például a SIN(X) kifejezés differenciálásával a COS(X) kifejezést kapjuk. A kifejezések tartalmazhatnak változókat, felhasználó által definiált függvényeket, egész, lebegőpontos vagy racionális konstansokat, valamint szimbolikus állandókat, mint a π, i és e, továbbá olyan függvényeket, mint SIN, COS, EXP stb. A kifejezéseket differenciálhatjuk, értéküket meghatározhatjuk. Egy kifejezésben egy változó vagy részkifejezés helyére egy másik kifejezést helyettesithetünk, két kifejezést egymással összehasonlithatunk.

A FORMAC alapvető koncepcióját Jean E. Sammet, az IBM munkatársa dolgozta ki 1962-ben [3]. A nyelv leirásának első teljes változata 1962 végén készült el; a gépi implementálás nem sokkal ezután kezdődött meg. A FORMAC ezen változata az IBM 7090/94 gépre készült az IBM IBSYS-IBJOB monitor alatt a FORTRAN IV nyelv egy kiterjesztéseként [5], [6], [7]. Az elkövetkező 18 hónap során a FORMAC nyelvet mint egy kisérletet kezelték, az IBM cégen kivül való felhasználásáról szó sem volt.

Az IBM 360-as sorozatu gépein való fejlesztési munkák csak 1964 őszén kezdődtek meg, azzal a szándékkal, hogy az előzőnél szélesebb felhasználói lehetőségeket hozzanak létre, és az uj nyelvet elsősorban a PL/I-hez, és ne a FORTRAN-hoz kapcsolják.

A PL/I-FORMAC rendszer 1967 novemberében készült el [1], [2]. Az alapvető cél az volt, hogy egy általánosan felhasználható, nagyon praktikus rendszert hozzanak létre. Mint azt látni fogjuk, bizonyos egyszerüsitésekre szükség volt annak érdekében, hogy ezt a célt ésszerü időn belül elérhessék. Az első döntés az volt, hogy a FORMAC-nak feltétlenül egy általánosan felhasználható rendszernek kell lennie, és nem egyedi feladatokat elvégző szubrutinok halmazának. Itt érdemes megemliteni, hogy a FORMAC-ot elsősorban gyakorlatiasnak és könnyen felhasználhatónak készitették. Ez a célkitüzés bizonyos mértékig a matematikai elegancia rovására érvényesült.

A FORMAC rendszer fejlesztését R. Tobey és társai 1969-ben le-

zárták.

A 60-as évek második felében az uttörő FORMAC mellett sorra jelentek meg és terjedtek el az ujabb formulamanipulációs nyelvek /REDUCE2 [13], ALTRAN [14] stb/, amelyek sok szempontból gazdagabb lehetőségeket nyujtottak. Azért, hogy a FORMAC nyelv ezzel a fejlődéssel lépést tartson, Knut Bahr /Darmstadt, NSzK/ 1973-ban átdolgozta és modernizálta [8]. Az uj FORMAC73 rendszer néhány, a nyelv erejét növelő lehetőséggel bővült, és az egész rendszer működése gyorsabb, hatékonyabb lett.

A FORMAC-nak a legalapvetőbb és legfontosabb lehetősége, hogy a matematikai kifejezéseket szimbolikus kifejezésekként képes kezelni. Ez a tulajdonság legjobban a FORMAC nem-numerikus lehetőségeinek és a PL/I numerikus módszereinek egybevetésével illusztrálható. Például tekintsük az alábbi PL/I programrészletet:

```
A = 4.6;
B = 2;
C = A/B+2.8;
```

Ennek végrehajtása után a C PL/I változó az 5.1 értéket kapja. Nézzük ezzel szemben a következő FORMAC programot:

```
LET(A = X+Y**2;
    B = 2;
    C = A/B+2.8);
```

Ennek végrehajtása a C FORMAC változóban az $(X+Y^2)/2+2.8$ kifejezést eredményezi. A C változó értéke tehát egy kifejezés.

A PL/I-FORMAC nyelv a PL/I nyelvnek egy kiterjesztése. Ez azt jelenti, hogy a FORMAC nyelv ugy jött létre, hogy a PL/I programozási nyelvet olyan utasitásokkal bővitették, amelyek matematikai kifejezések szimbolikus kezelésére alkalmasak.

Egy FORMAC program tehát FORMAC utasitásokból és PL/I utasitásokból áll.

A FORMAC programok forditását egy előfeldolgozó program, un. preprocesszor végzi. Ez a FORMAC utasitásokat PL/I utasitásokra forditja le. Az igy létrejött programot ezután a PL/I forditóprogrammal kell leforditani.

A következőkben rövid bevezetést adunk a FORMAC nyelv használatába. Célunk, hogy az olvasó általános áttekintést szerezzen

a nyelvről, és egyszerübb programokat össze tudjon állitani. A FORMAC részletesebb ismertetése az [1] és [2] kézikönyvekben található.

A megfelelő helyeken kitérünk a FORMAC73 rendszernek az eredeti rendszertől való eltéréseire. Ezek leirását Knut Bahr [8] cikkében találhatjuk meg.

A PL/I nyelvet ismertnek tételezzük fel [15], [16]. Az 1. fejezetben a FORMAC nyelv utasitásait ismertetjük. A 2. fejezetben foglalkozunk a FORMAC és a PL/I közötti kapcsolattal, kölcsönhatásokkal. A 3. fejezetben a FORMAC programban használható ON-feltételeket irjuk le. A 4. fejezetben áttekintjük a FORMAC program forditásának és végrehajtásának menetét. Végül az 5. fejezetben két teljes FORMAC programot mutatunk be a futtatás eredményével együtt. A függelékek tartalmazzák a FORMAC nyelvben érvényes korlátozásokat, a FORMAC rendszer hibajelzéseit, valamint a FORMAC73 rendszerben az eredeti rendszerhez képest található eltérések /bővitések, módositások/ összefoglalását.

1. A nyelv elemei

Ebben a fejezetben a FORMAC nyelv elemeit ismertetjük. Előre bocsátjuk azt, hogy a FORMAC nyelv a PL/I nyelv kiterjesztése. Igy minden FORMAC program egyszersmind egy szabályszerü PL/I program, amely azonban FORMAC utasitásokat is tartalmaz. A FORMAC programok irására tehát ugyanazok az általános szabályok érvényesek, mint a PL/I programok irására. Néhány ilyen szabály:

- a programnak a

  programnév: PROCEDURE OPTIONS(MAIN);

  utasitással kell kezdődnie, ahol "programnév" egy tetszőleges, legfeljebb 8 karakter hosszuságu, a PL/I-ben megengedett azonositó,
- az utasitásokat a 2-72. karakterpoziciókra irjuk,
- a megjegyzéseket /kommentárok/ a "/*" és "*/" jelek közé foglalva irjuk. Pl:

  /*EZ EGY MEGJEGYZÉS*/
- az utasitások végét pontosvessző /;/ jelöli.

A következőkben a definiciókban nagybetükkel irjuk mindazt, amit a programozónak is mindig ugyanigy kell irnia: a FORMAC kulcsszavakat, a standard függények neveit stb. Kisbetükkel jelöljük azokat a változókat, kifejezéseket, amelyek helyére a programozónak az utasitás, függvény stb használatakor a megfelelő konkrét változókat stb kell behelyettesitenie.

1.1 A FORMAC értékadó utasitás

A FORMAC nyelvben az értékadó utasitás formája

var = expr

ahol "var" egy FORMAC változó neve, "expr" pedig egy FORMAC kifejezés. Az utasitás végrehajtásának hatására a "var" változó felveszi a jobboldalon álló kifejezés aktuális értékét.

A FORMAC értékadó utasitást a PL/I nyelv értékadó utasitásától formailag a "LET" kulcsszó különbözteti meg. A "LET" kulcsszó után az értékadó utasitást zárójelek közé foglalva irjuk. Az

utasitás teljes formája tehát

```
LET(var = expr);
```

Példák:

```
LET(A = B+C);
LET(X = A+3*A**2);
```

Egyetlen "LET" kulcsszóval több FORMAC értékadó utasitást is leirhatunk. A fenti példát igy is irhatjuk:

```
LET(A = B+C;
    X = A+3*A**2);
```

Az egy "LET" kulcsszóhoz tartozó értékadó utasitások mindegyike után pontosvesszőt kell tenni, kivéve a legutolsót: ennek végét a bezáró zárójel jelöli. /Vigyázzunk az értékadó utasitások leirásánál! Gyakori hiba, hogy az értékadó utasitás vagy utasitások végén a bezáró zárójelet elfelejtjük kitenni!/

A FORMAC értékadó utasitás a PL/I nyelv értékadó utasitásának algebrai ekvivalense.

A PL/I nyelvben az értékadás "=" jelének hatására a baloldalon álló változóba egy számérték /vagy karakterfüzér, bitfüzér stb/ kerül. A FORMAC értékadó utasitás hatására a baloldalon álló változó értéke mindig egy kifejezés lesz. Az utasitás jobboldalán szereplő kifejezés végrehajtás közben kiértékelődik: ez azt jelenti, hogy elvégződnek rajta a szükséges átalakitások /pl. egyszerüsités, behelyettesités stb/. A kiértékelés eredménye általában egy uj kifejezés: a baloldalon álló változónak ez lesz az értéke.

Az előző példában a

```
LET(A = B+C);
```

utasitás végrehajtása után az A változó értéke a B+C kifejezés. Ezután a

```
LET(X = A+3*A**2);
```

utasitás végrehajtása után az X változó értéke a $B+C+3B^2+6B\ C+3C^2$ kifejezés. Az A változó helyére a kiértékelés során mindenütt a B+C kifejezés helyettesitődik.

A következő pontokban részletesen leirjuk a FORMAC nyelv alkotóelemeit, melyekből az értékadó utasitás felépül.

1.2 FORMAC konstansok

Négyféle FORMAC konstans létezik:

lebegőpontos,
egész,
racionális,
rendszerkonstans.

A lebegőpontos konstansok számtartománya kb. $5.4*10^{-79}$-től kb. $7.2*10^{75}$-ig terjed. A FORMAC rendszer un. kétszeres pontossággal ábrázolja őket /kb. 16 decimális jegy pontossággal/. Ez a PL/I nyelv FLOAT BINARY(53) adattipusának felel meg.

Egy egész konstanst legfeljebb 2295 darab decimális számjegygyel ábrázolhatunk, tehát gyakorlatilag tetszőleges hosszuságu egész számokkal dolgozhatunk.

A racionális számok a belső ábrázolásban a/b alakban ábrázolódnak, ahol a és b egész számok, és a rendszer a lehetséges egyszerüsitéseket elvégezte, azaz a és b relativ primszámok. Itt a és b szintén legfeljebb 2295 jegyü egész szám lehet.

A FORMAC nyelv egyik legfontosabb sajátossága az, hogy az egész és racionális számokat a rendszer ebben a formában tárolja, nem helyettesiti őket közelitő pontosságu lebegőpontos értékekkel. Igy az egész és racionális számokkal végzett müveletek mindig pontosan hajtódnak végre.

A rendszernek ez a sajátossága feltétlenül szükséges ahhoz, hogy a szimbolikus számitásoknál szokásos kérdésekre feleletet kereshessünk, például arra, hogy két kifejezés egyenlő-e egymással, vagy egy kifejezés egyenlő-e 0-val. Ahhoz, hogy ilyen kérdéseket vizsgálhassunk, szükséges, hogy a vizsgált kifejezésekben szereplő állandókat egzakt pontossággal ábrázoljuk. Igy például, ha sin(x) sorfejtésének első néhány tagját az

```
X-X**3/(2*3)+X**5/(2*3*4*5)-X**7/(2*3*4*5*6*7)
```

kifejezéssel adjuk meg, a FORMAC rendszer ezt az

```
X-(1/6)*X**3+(1/120)*X**5-(1/5040)*X**7
```

formában tárolja, nem pedig a közelitő pontosságu

```
X-0.1666667*X**3+0.008333333*X**5-0.000198413*X**7
```

alakban.

A rendszerkonstansok jelölésére három fenntartott név szolgál:

#E a természetes logaritmus alapszámát jelöli (e);
#P a π-t reprezentálja;
#I a képzetes egységet jelöli /#I=$\sqrt{-1}$/.

1.3 FORMAC változók

A FORMAC változókat két tipusra osztjuk:

atomi és
kijelölt változókra.

Egy atomi változó vagy más szóval atom, olyan változó, amely még nem kapott értéket, vagyis nem szerepelt egy végrehajtott FORMAC értékadó utasitás baloldalán. Mint ilyen, egy atom önmagát jelöli. Egy kijelölt változó olyan változó, amely már értéket kapott, tehát nem önmagát jelöli, hanem egy kifejezést tartalmaz. Ezt a kifejezést a változó aktuális értékének nevezzük. Például a

```
LET(B = X*Y);
LET(A = B+C);
```

utasitások végrehajtása után B és A kijelölt változók, X, Y és C pedig atomok, feltéve, hogy ezt megelőzően nem kaptak a programban értéket. B aktuális értéke X*Y. A második utasitásban szereplő kifejezés kiértékelésekor a B változó helyére ez a kifejezés helyettesitődik be, igy A aktuális értéke X*Y+C lesz.

Felhivjuk a figyelmet arra, hogy a FORMAC nyelv megkülönbözteti a FORMAC változókat és a PL/I változókat. A programban szerepelhet olyan FORMAC változó, melynek neve azonos egy PL/I változó nevével: a rendszer azonban ezeket két külön változóként kezeli. Például az

```
A = 5;
LET(A = 6; B = X-Y);
B = A;
```

utasitások végrehajtása után az A és B PL/I változók értéke 5-tel lesz egyenlő, mig az A FORMAC változó értéke 6-tal, a B FORMAC változó értéke X-Y-nal lesz egyenlő.

A programozónak azonban lehetősége van arra, hogy egy FORMAC utasitásban PL/I változót használjon, vagy egy FORMAC változó értékét egy PL/I változónak átadja. Ezeket a lehetőségeket a 2. fe-

jezetben ismertetjük.

A FORMAC változókat nem deklaráljuk.

A FORMAC változóknak legfeljebb 4 indexe lehet. Az index egész tipusu FORMAC konstans, vagy egész értékü FORMAC kifejezés lehet. Az index számértéke a -31622 és 31622 határok közé eshet. Az indexes FORMAC változókat sem deklaráljuk.

A FORMAC változók neve tetszőleges, legfeljebb 8 karakter hosszuságu alfanumerikus jelsorozat lehet. A "_" /aláhuzás/ karakter, amely PL/I változók nevében előfordulhat, FORMAC változók nevében nem szerepelhet.

Felhivjuk a figyelmet arra, hogy a FORMAC kulcsszavak védettek: tehát LET, SIN, FAC, CONVERT stb nem használható FORMAC változónévként. Kerüljük továbbá - esetleges névütközések elkerülése céljából - a "DEN" karakterekkel kezdődő neveket is, mivel a FORMAC rendszer minden belső rutinjának a neve igy kezdődik.

1.4 FORMAC kifejezések

A FORMAC kifejezéseket formailag azonos módon irjuk, mint a PL/I kifejezéseket. A lényeges különbség az, hogy a FORMAC kifejezés FORMAC változókból és konstansokból épül fel, és a kifejezést a rendszer szimbolikusan kezeli. A FORMAC változókat és konstansokat a +, -, *, /, ** müveletekkel kapcsolhatjuk össze.

A végrehajtási sorrendet vagy explicit módon zárójelekkel határozhatjuk meg, vagy implicite a müveletek szokásos hierarchiája szerint: legnagyobb prioritásu a hatványozás, majd az osztás és szorzás azonos prioritással, és végül az összeadás és a kivonás müveletei kerülnek végrehajtásra. Azonos prioritásu müveletek elvégzésének sorrendjét a "balról jobbra" szabály határozza meg.

A FORMAC kifejezések függvényhivásokat is tartalmazhatnak. A függvények különböző tipusait az 1.5 pontban ismertetjük.

Egy FORMAC értékadó utasitás végrehajtásakor, mint azt az 1.1 pontban mondottuk, a jobboldalon álló kifejezés kiértékelődik. A kiértékelés során minden, a kifejezésben szereplő kijelölt változó helyére annak aktuális értéke helyettesitődik. Ezután a kifejezésen a FORMAC rendszer még további átalakitásokat is végez, azt egyszerübb alakra hozza. Az ennek során alkalmazott egyszerüsitési

szabályokat az 1.8 pontban ismertetjük.

A programozó bizonyos mértékig irányithatja azt, hogy a rendszer milyen átalakitásokat végezzen el a kifejezésen az egyszerüsités során. Például: a kifejezésekben szereplő törteket hozza-e közös nevezőre, vagy: szorzatkifejezésekre alkalmazza-e a disztributiv törtvényt a rendszer, azaz bontsa-e fel a zárójeleket. A programozó az OPTSET utasutás segitségével irhatja elő, hogy a rendszerben az egyszerüsitéskor milyen FORMAC opció legyen érvényben.

Egy példa:

ha az

```
OPTSET(EXPND);
```

utasitás érvényes, akkor például a

```
LET (X =(A+2)*(P+Q));
```

utasitás eredményében a FORMAC rendszer felbontja a zárójeleket és az eredmény

```
X←A P+A Q+2 P+2 Q
```

lesz. Ha az

```
OPTSET(NOEXPND);
```

utasitás érvényes, a kifejezés ki nem fejtett szorzatalakban marad.

Az OPTSET utasitásban megadható opciókat az 1.9 pontban ismertetjük. Addig is az egyes FORMAC függvények ismertetésekor utalunk arra, ha a függvény hivásának eredménye függ egy FORMAC opciótól.

Az, hogy hogyan használhatunk a FORMAC kifejezésekben PL/I változókat is, a 2.1 pontban irjuk le.

1.5 FORMAC függvények

A FORMAC függvények a következő négy kategóriába sorolhatók:

1. PL/I tipusu függvények,
2. egészértékü függvények,
3. a felhasználó által definiált függvények,
4. függvényváltozók.

1.5.1 PL/I tipusu függvények

A legfontosabb PL/I tipusu függvények a következők:

SIN	ATAN	LOG10
COS	ATANH	LOG2
SINH	ERF	LOG
COSH		

Ezek a PL/I nyelvben definiált függvények a FORMAC utasitásokban is használhatók. Hivásuk eredménye az OPTSET utasitásban megadható TRANS, illetve NOTRANS opciótól függ /l. még az 1.9 pontot/.

Ha a TRANS opció érvényes, a függvény hivásának eredménye mindig egy lebegőpontos szám. Bizonyos transzformációk ilyenkor automatikusan végrehajtódnak, mint pl. SIN(P/6)←1/2.

Ha a NOTRANS opciót adjuk meg, akkor a függvények szimbolikus mennyiségek maradnak még abban az esetben is, ha argumentumuk egy szám. Például

```
LET (A = LOG(X**2)+SIN(3/4));
```

eredménye

A←LOG(X^2)+0.68163876,

ha a TRANS opció érvényes, és

A←LOG(X^2)+SIN(3/4),

ha a TRANS opció nem érvényes.

1.5.2 Egészértékü függvények

Egészértékü függvények a FAC, COMB és STEP.

FAC(n) = n! /n faktoriális/

$$COMB(n,n_1,\ldots,n_k) = \frac{n(n-1)\ldots(n-n_1-\ldots-n_k+1)}{n_1!\ldots n_k!} \quad (\Sigma n_i \leq n)$$

A FAC és COMB függvények hivásának eredménye is függ egy, az OPTSET utasitásban előirható opciótól, az INT opciótól /l. az 1.9 pontot/.

Az INT opció megadásakor a FAC és a COMB függvény egy egész értéket ad az argumentum tipusától függetlenül, ha az argumentum nem negativ. Negativ argumentumok nem megengedettek. Ha a NOINT

opció érvényes, akkor a FORMAC rendszer a FAC és a COMB függvényeket argumentum értékükkel együtt szimbolikus mennyiségekként kezeli.

Tehát pl. ha az INT opció érvényes, a

```
LET (A = FAC(6));
```

utasitás eredménye: A←720.

Ha azonban a NOINT opció érvényes, az utasitás eredményét a rendszer az A←FAC(6) szimbolikus alakban fogja tárolni.

A STEP függvénynek három argumentuma van.

$$STEP(expr_1,expr_2,expr_3)$$

a következő értékeket veszi fel /$expr_1$, $expr_2$ és $expr_3$ FORMAC kifejezések/:

$$STEP(expr_1,expr_2,expr_3) = \begin{cases} 1, & \text{ha } expr_1 \leq expr_2 < expr_3, \\ & \text{vagy } expr_1 = expr_2 = expr_3, \\ 0 & \text{egyébként.} \end{cases}$$

Ha az $expr_1$, $expr_2$, $expr_3$ kifejezések közül bármelyiknek a kiértékelése nem-numerikus értéket szolgáltat, akkor a STEP függvény a kiértékelt argumentumaival együtt szimbolikusan kezelt függvény marad.

Például a

```
LET (I = 10;
     P = STEP(I-1,I,I+1);
     Q = STEP(X-Y,X,X+Y));
```

utasitások végrehajtása után a P FORMAC változó értéke 1 lesz, mert az I FORMAC változónak numerikus értéke van, a Q FORMAC változó azonban a STEP(X-Y,X,X+Y) szimbolikus értéket tartalmazza.

1.5.3 A felhasználó által definiált függvények

A FORMAC nyelvben a felhasználó maga is definiálhat ujabb függvényeket.

Az

```
FNC(függvénynév) = expr($(1),$(2),...,$(n))
```

utasitás a "függvénynév" nevü függvényt, mint n-változós függvényt definiálja.

Az expr kifejezésben $(i) az i-edik formális paramétert /a függvény i-edik argumentumát/ jelöli /i=1,2,...,n/.

Például a

```
LET(FNC(F) = 3*COS($(1)**2+$(2))+1);
```

utasitás az F függvényt kétváltozós függvényként definiálja. Ez a matematikában szokásos irásmód szerint az

$$F(arg_1,arg_2) = 3\cos(arg_1^2+arg_2)+1$$

definiciónak felel meg, ahol arg_1 és arg_2 az F függvény két változóját jelöli.

Az F függvényre való minden későbbi hivatkozáskor $F(arg_1,arg_2)$ helyére a definició jobboldalán megadott kifejezés helyettesitődik be, a megfelelő aktuális paraméterekkel.

Ha az F függvényt a fenti FORMAC utasitással definiáljuk, a

```
LET (X = B+C;
     Y = F(X+4,3)+5);
```

utasitások eredményeként az Y FORMAC változóban a

$$Y \leftarrow 3\cos((B+C+4)^2+3)+6$$

kifejezést kapjuk. Itt X+4 felel meg az F függvény első argumentumának, 3 pedig a második argumentumnak.

Megjegyzések:

1. A felhasználó által definiált függvény neve nem lehet azonos egy FORMAC változó nevével.

2. A függvényt az előtt kell definiálni, mielőtt egy FORMAC kifejezésben hivatkoznánk rá.

3. Korábban definiált függvények is szerepelhetnek FNC utasitások jobboldalán.

4. A függvények nem definiálhatók ujra.

Az átdolgozott FORMAC73 rendszer a függvények formális paramétereinek kiértékelését "késleltetve", a függvény hivásakor végzi el, mig az eredeti FORMAC rendszer a függvény definiálásakor azonnal elvégezte a kiértékelést. Ezzel a változtatással a FORMAC73 rendszerben általánosabb és hajlékonyabb függvénydefiníciókat lehet megadni. Például:

```
LET(FNC(FV) = $(1)+ARG(1,$(2)));
```

Az itt használt ARG(n,z) függvény a z kifejezés n-edik ar-

gumentumát állitja elő /definicióját lásd az 1.8.5 pontban/. Az eredeti FORMAC rendszer a függvénydefinició feldolgozásakor az ARG(1,$(2)) kifejezést 0-val tette egyenlővé, mivel a $(2) tagot atomnak tekintette.

A FORMAC73 rendszer a jobboldalon álló kifejezést csak az FV függvény hivásakor értékeli ki, amikor az aktuális argumentumokat már ismerjük. Igy a

```
LET (Z = FV(A,SIN(A+B)));
```

utasitás eredménye az eredeti FORMAC rendszerben

```
Z ← A
```

a FORMAC73 rendszerben viszont

```
Z ← 2 A+B.
```

FV második argumentuma ugyanis a SIN(A+B) kifejezés, ennek első /és egyetlen/ argumentuma pedig A+B.

1.5.4 Függvényváltozók

FORMAC kifejezésekben olyan függvényeket is használhatunk, amelyeket előzőleg nem specifikáltunk explicit módon formulával, az előző pontban leirt

```
LET (FNC(név) = ...);
```

utasitás segitségével.

Ezeket a specifikálatlan függvényeket az un. függvényváltozók segitségével jelölhetjük ki.

A függvényváltozó neve a FORMAC kifejezésben argumentumlistájával együtt szerepel. A függvényváltozó neve után pont /./ áll, az argumentumok tetszőleges FORMAC kifejezések lehetnek, például:

```
LET (Y = F.(A+B,A-B));
```

Itt F a függvényváltozó neve, amely után pontot teszünk. A+B és A-B a függvény argumentumai.

A függvényváltozók kifejezésekben szerepelhetnek, formálisan differenciálhatjuk őket /l. az 1.8.3 pontot/.

A függvényváltozókat a későbbiekben helyettesithetjük egy, már ismert kifejezéssel vagy függvénnyel /az EVAL rutin segitségével, l. az 1.8.2 pontot/.

1.6 Listák /láncok/ létrehozása /CHAIN/

FORMAC változók értéke lehet egy FORMAC kifejezésekből alkotott lista vagy lánc is. Az ilyen értékadásokat a CHAIN müvelet segitségével végezhetjük el. A CHAIN müveletnek tetszőleges számu argumentuma lehet:

CHAIN ($expr_1$,$expr_2$,...,$expr_n$)

ahol $expr_1$,$expr_2$,...,$expr_n$ FORMAC kifejezések. A CHAIN müvelet ezeket a kifejezéseket - kiértékelésük után - egy láncba /listára/ füzi fel.

Ez a müvelet jól használható arra, hogy FORMAC függvények és rutinok argumentumlistáját definiáljuk. Például:

```
LET(X = A+3*SIN(B);
    Y = CHAIN(A,X+A,B+7,4));
```

azt eredményezi, hogy Y értéke az (A,2 A+3 SIN(B),B+7,4) lista lesz.

Ha ezután a

```
LET (Z = F.(Y));
```

utasitást hajtjuk végre, ennek hatására Z értéke

```
Z←F.(A,2 A+3 SIN(B),B+7,4)
```

lesz.

Itt feltételezzük, hogy F.(...) négyváltozós függvényváltozó.

A CHAIN müvelet segitségével tehát itt az F specifikálatlan függvény számára összeállitott argumentumlistát az Y változóban helyeztük el, és ennek a változónak az értékét adjuk át argumentumként az F függvénynek.

Megjegyezzük, hogy a CHAIN müvelet argumentumai azonnal kiértékelődnek. Például a fenti példában X helyére behelyettesitődött az A+3*SIN(B) kifejezés.

A CHAIN kulcsszót el is hagyhatjuk. A fenti példát a

```
LET (X = A+3*SIN(B);
     Y = (A,X+A,B+7,4));
```

formában is irhatjuk.

1.7 FORMAC programok outputja

A PRINT_OUT utasitást FORMAC változók értékének kinyomtatására használjuk. Például:

```
LET (X = A**5+SIN(3*A)**2+2/3);
PRINT_OUT(X);
```

a következő nyomtatott sort fogja eredményezni:

$$X = A^5+SIN^2(3A)+2/3$$

A PRINT_OUT a kitevőket egy soremeléssel nyomtatja és elhagyja a szorzás jelölésére szolgáló csillagot /*/. Igy az output formája a szokásos matematikai jelöléshez közelebb áll.

Megjegyzések:

1. A PRINT_OUT utasitás tetszőleges számu változót tartalmazhat argumentumként, melyeket pontosvessző választ el egymástól, például:

```
PRINT_OUT(X;Y;Z);
```

Ez az utasitás mindegyik változó nevét és értékét uj sorban nyomtatja ki.

2. Egy atom értékét nyomtatáskor az üres jelsorozat reprezentálja. Például, ha X egy atom, akkor a PRINT_OUT(X); utasitás végrehajtásának eredménye:

```
X =
```

3. A lebegőpontos számokat a PRINT_OUT utasitás legfeljebb 9 értékes jeggyel és tizedesponttal nyomtatja ki. Megjegyezzük, hogy a lebegőpontos müveleteket ennél nagyobb pontossággal végzi a FORMAC /kétszeres pontossággal, l. az 1.2 pontot/. Az "E" kitevőjelet használja a rendszer olyan számok nyomtatásakor, amelyek a 10^{-2}-10^{+6} számtartományon kivül esnek. Az egész és racionális számokat teljes egészükben kinyomtatja. Az egész számok legfeljebb 2295 számjegyből állhatnak. A racionális számokat a PRINT_OUT két egész szám hányadosaként nyomtatja ki, például:

```
A = 5/3
```

Az OPTSET(PROPER) opció hatására /l. 1.8.7 pont/ a fenti A változó az

```
A = 1+2/3
```

formában /valódi tört formájában/ jelenik meg.

4. A PRINT_OUT utasitás argumentumlistáján szerepelhet érték-

adó utasitás is. Ekkor a LET kulcsszót nem irjuk ki. Például:

```
PRINT_OUT(A=B+C; D=SIN(X));
```

egyenértékü a

```
LET (A = B+C;
     D = SIN(X));
PRINT_OUT(A;D);
```

utasitássorozattal.

1.8 FORMAC rutinok

A FORMAC nyelvben a FORMAC rutinok arra szolgálnak, hogy a kifejezéseken különböző átalakitásokat végezzünk.

A FORMAC rutinok függvények, melyeknek értéke mindig egy FORMAC kifejezés.

A FORMAC rutinok argumentumait - amelyek általában FORMAC kifejezések - a rendszer mindig még a rutin hivása előtt kiértékeli.

A FORMAC rutinok a következő csoportokba sorolhatók:

- kifejezések felhasználó által irányitott egyszerüsitése /MULT, DIST, EXPAND, CODEM, FRACTN/;
- kifejezéseknek más kifejezésekbe való behelyettesitése /EVAL, REPLACE/;
- differenciálás /DERIV, DIFF, DRV/;
- összehasonlitás /IDENT/;
- kifejezések részekre bontása /COEFF, HIGHPOW, LOWPOW, NUM, DENOM, LOP, NARGS, ARG/;
- gazdálkodás a memóriával /SAVE, ATOMIZE/.

1.8.1 A felhasználó által irányitott egyszerüsités

A kifejezések egyszerübb alakra hozatala /egyszerüsitése/ minden formulamanipulációs nyelvben alapvető fontosságu és bonyolult kérdés. A fő problémát az okozza, hogy nem lehet egyértelmü módszert találni annak eldöntésére, hogy egy kifejezésnek melyik a "legegyszerübb" alakja.

Azt általában elfogadjuk, hogy a

```
3*X
```

alak egyszerübb, mint

```
X+X+X.
```

Azt már nem ilyen egyszerü eldönteni, hogy az

```
(X+1)**2
```

alak egyszerübb-e, vagy az

```
X**2+2*X+1
```

alak. A döntés függhet attól, hogy a kifejezésekkel milyen további müveleteket végzünk.

Egyes formulamanipulációs rendszerek az utóbbi /kifejtett/ alakot tekintik egyszerübbnek, és minden olyan kifejezést, amely egy polinom valamilyen egész kitevőjü hatványa, automatikusan kifejtett alakra hoznak.

Az ilyen rendszerek használatakor nehéz helyzetbe kerülünk, ha például az

```
(X+1)**1000
```

kifejezéssel kell dolgoznunk, amelynek kifejtett alakja

```
X**1000+1000*X**999+...+1000*X+1
```

1001 /!/ tagból áll.

A különböző formulamanipulációs nyelvek kidolgozóinak dönteniök kellett, hogy az egyszerüsitést a rendszer automatikusan elvégezze-e, vagy pedig a felhasználó szabadon dönthessen az alkalmazandó szabályokról. Erről a kérdéskörről jó áttekintést ad J. Moses [12].

A FORMAC rendszer kidolgozói "középutas" megoldást választottak, és a rendszerbe kétféle egyszerüsitési lehetőséget épitettek be:

- egy rutint /AUTomatic SIMplification, AUTSIM/, amely minden kifejezésen automatikusan elvégez bizonyos "kézenfekvő" egyszerüsitéseket,
- néhány további rutint, amelyeknek segitségével a felhasználó maga dönti el, hogy milyen alakra hozza a rendszer a kifejezést.

Az AUTSIM rutin a programozó beavatkozása nélkül, mintegy a programozó "háta mögött" minden olyan FORMAC kifejezésen végrehajtódik, amely egy FORMAC értékadó utasitás jobboldalán szerepel. A rutin azokat az átalakitásokat végzi el, amelyeket a "papiron ceruzával" való számolásnál is rendszerint elvégzünk.

Mint azt a kifejezések kiértékelésénél /1.4 pont/ már mondottuk, a programozó az OPTSET utasitásban előirható FORMAC op-

ciók segitségével bizonyos mértékig az AUTSIM rutin müködését is vezérelheti: előirhatja, hogy a rutin milyen átalakitásokat végezzen el.

Az alábbiakban összefoglaljuk az AUTSIM rutin által használt legfontosabb egyszerüsitési szabályokat. A továbbiakban az A, B és C betük FORMAC változókat jelentenek, n pedig egész számot.

$$A^0 \rightarrow 1$$

$$A^1 \rightarrow A$$

$$(-A)^n \rightarrow \begin{cases} A^n & \text{ha n páros} \\ -A^n & \text{ha n páratlan} \end{cases}$$

$$-(A+B+C) \rightarrow (-A)+(-B)+(-C)$$

$$-(-A) \rightarrow A$$

$$A+0 \rightarrow A$$

$$A+2A \rightarrow 3A$$

$$A*1 \rightarrow A$$

$$0^0 \rightarrow \text{hiba}$$

$$SIN(-A) \rightarrow -SIN(A)$$

$$\left.\begin{array}{l} SIN(0) \rightarrow 0 \\ SIN(\#P) \rightarrow 0 \end{array}\right\} \quad \text{TRANS opciótól függően, 1. az 1.9 pontot}$$

Ezenkivül az AUTSIM lexikografikus sorrendbe helyezi a kifejezésekben előforduló változóneveket, tehát például a következő szabályokat alkalmazza:

$$B+A \rightarrow A+B$$

$$B*A \rightarrow A*B$$

A szabályokat részletesebben ismerteti az [1] kézikönyv /136-147. oldal/. Az AUTSIM rutin kidolgozásakor követett elveket R. G. Tobey és társainak cikke tárgyalja [4].

Az AUTSIM rutin egyes funkcióit a felhasználó az OPTSET utasitás következő paramétereivel /opcióival/ is vezérelheti /az alapértelmezést aláhuzással jelöltük/:

TRANS NOTRANS — transzcendens függvények /SIN, LOG stb/ értékének kiszámitását vezérlik, ha a függvény argumentuma egy szám. Ha TRANS érvényes, az eredmény egy számérték, ha NOTRANS, akkor szimbolikus kifejezés marad;

INT NOINT — egész tipusu FORMAC függvények /FAC, COMB, STEP/ értékének kiszámitását vezérlik, ha a

függvény argumentuma/i/ szám/ok/. Ha INT érvényes, az eredmény egy számérték, ha NOINT érvényes, akkor az eredmény egy szimbolikus kifejezés;

EXPND NOEXPND — azt határozzák meg, hogy egy kifejezés kiértékelésekor alkalmazza-e a rendszer a disztributiv törvényt és a polinomiális tételt /zárójelek felbontása/. Az AUTSIM szempontjából ez a legfontosabb opció. E szerint az AUTSIM rutin csak akkor végzi el automatikusan a zárójelek felbontását, ha az EXPND opció érvényes.

A FORMAC opciók részletes leirását lásd az 1.9 pontban!

Nem végzi el az AUTSIM rutin a következő átalakitásokat /akkor sem, ha EXPND érvényes/:

- kifejezés nevezőjének kifejtését: pl. ha EXPND érvényes, akkor (A+B)**2 → A**2+2*A*B+B**2, de 1/(A+B)**2 változatlan alakban marad;
- törtkifejezések közös nevezőre hozását: pl. A/B+C/D változatlan marad, nem alakitódik át az (A*D+B*C)/(B*D) alakra.

Ezeket az utóbb felsorolt átalakitásokat a FORMAC rendszer csak a programozó utasitására végzi el.

A FORMAC nyelvet, azért mert az egyszerüsités kérdésében aránylag sok döntési lehetőséget hagy meg a programozónak, J.Moses idézett cikkében a "liberális" rendszerek közé sorolja.

A továbbiakban azokat a rutinokat ismertetjük, amelyek segitségével a felhasználó irányithatja a kifejezések egyszerüsitését.

Ezeket a rutinokat a következő osztályokba sorolhatjuk:

- a kifejezések kifejtésére szolgáló rutinok a MULT, DIST és az EXPAND, amelyek az asszociativ és/vagy disztributiv törvényeket használják;
- a legkisebb közös többszörös rutinok, a CODEM és a FRACTN, amelyek a kifejezéseket közös nevezőre hozott (a/b) alakban adják vissza;
- a kifejezések legnagyobb közös tényezőjét előállitó GCF és GCFOUT rutinok /ezeket Knut Bahr illesztette az átdolgozás során a FORMAC73 rendszerhez [8]/.

MULT(expr) - az expr kifejezésben szereplő minden olyan összeget, amely pozitiv egész kitevőre van emelve, a polinomiális tételt felhasználva kifejtett alakra hoz; a kombinatorikus kifejezéseket a felhasználó választása szerint vagy kiszámitja /ha az INT opció érvényes/, vagy szimbolikus formában hagyja /ha a NOINT opció érvényes/;

DIST(expr) - a disztributiv törvényt használja az expr kifejezésben szereplő összegek szorzatainak kifejtésére /a zárójelek felbontására/; a DIST nem használja a polinomiális tételt és igy az egész kitevőre emelt összegeket változatlanul hagyja;

EXPAND(expr) - mind a polinomiális tételt, mind a disztributiv törvényt alkalmazza az expr-ben szereplő részkifejezésekre, és igy egy teljesen kifejtett eredményt ad a megfelelő tagok összevonásával.

Példák és megjegyzések:

1.
```
LET (X = (A+B)*(A-B)**2;
     Y = MULT(X));
```

Eredmény:

$Y \leftarrow (A+B)(-2A\,B + A^2 + B^2)$

2.
```
OPTSET (NOINT);
LET (X = 5*(A+B)**3;
     Y = MULT(X)+A**5);
```

Eredmény:

$Y \leftarrow 5(COMB(3,1)B^2A+COMB(3,2)B\,A^2+A^3+B^3)+A^5.$

Mivel a NOINT opció érvényes, a COMB(3,1) és COMB(3,2) függvények helyére nem helyettesitődik be a megfelelő számérték.

3.
```
LET (P = DIST(C*(A+B));
     Q = DIST((A+B)/C));
```

Eredmények:

$P \leftarrow A\,C + B\,C,$

$Q \leftarrow A/C + B/C.$

4.
```
LET (Y = (X-A)**2+2*(X-A)**4+3*(X-A)**6;
     Z = DIST(2*(X-A)**3*Y));
```

Eredmény:

$$Z \leftarrow 2(X-A)^5+4(X-A)^7+6(X-A)^9$$

5.
```
LET (X = A*(COS(B*(C+D))+A);
     Y = 3*(A-2)**2;
     Z = EXPAND(X-Y));
```

Eredmény:

$$Z \leftarrow 12A + A\ COS(B\ C + B\ D) - 2A^2 - 12.$$

A MULT, DIST és EXPAND nem módositja a teljesen faktorizált alakban felirt nevezőket. Például:

```
LET (X = 3*(Q+7)/(P*(Q+R)**3);
     X = EXPAND(X));
```

eredményeképpen

$$X \leftarrow \frac{3Q}{P(Q+R)^3} + \frac{21}{P(Q+R)^3}$$

lesz.

A következő csoportba tartozó rutinokkal /CODEM és FRACTN/ törtkifejezéseket hozhatunk közös nevezőre.

CODEM(expr) - az expr kifejezést az a/b alakra hozza, ha ez lehetséges /közös nevezőre hoz/. A részkifejezéseket is közös nevezőre hozza.

FRACTN(expr) - ugyanaz, mint a CODEM, kivéve, hogy csak a legkülső szinten végzi el a közös nevezőre hozatalt: a részkifejezéseket már nem hozza közös nevezőre.

Például:

6.
```
LET (X = A/(B+C/D)+E/F;
     Y = CODEM(X);
     Z = FRACTN(X));
```

Eredmények:

$$Y \leftarrow \frac{F\ D\ A + (C + D\ B)\ E}{(C + D\ B)\ F}$$

$$Z \leftarrow \frac{F\ A + (B + C/D)\ E}{(B + C/D)\ F}$$

tehát a FRACTN a számlálóban, illetve a nevezőben szereplő kifejezéseket már nem hozza közös nevezőre.

A rutinok harmadik csoportja /GCF és GCFOUT/ kifejezések közös tényezőjének meghatározására, illetve a közös tényező kiemelésére szolgál. Ezeket a rutinokat csak az átdolgozott FORMAC73

rendszer tartalmazza.

A GCF /Greatest Common Factor/ függvény kétféleképpen hivható: ha egyetlen argumentuma van, és az egy összeg, akkor a GCF függvény a tagok legnagyobb közös tényezőjét adja. Ha a GCF-nek több argumentuma van, akkor a függvény az argumentumok legnagyobb közös tényezőjét adja eredményül.

Példák:

7. LET (Z = GCF(A+A**2+A**3));

Eredmény:

Z ← A

8. LET (Z = GCF(A*(X+Y)+B*(X+Y)));

Eredmény:

Z ← X+Y

9. LET (Z = GCF(350,630));

Eredmény:

Z ← 70

10. LET (Z = GCF(A*B*C*D,B*C*D*E,C*D*E*F));

Eredmény:

Z ← C D

11. A

LET (Z = GCF(X+Y,A*X+A*Y+B*X+B*Y));

utasitás eredménye azonban

Z ← 1 /!/

Az X+Y közös tényezőt a GCF függvény már nem tudja kiemelni. Ebből látszik, hogy a GCF rutin csak a "nyilvánvaló" közös tényezőket találja meg, tehát nem tartalmaz egy általános algoritmust a legnagyobb közös osztó /GCD/ megkeresésére.

A GCFOUT függvény a GCF által megtalált legnagyobb közös tényezőt kiemeli a kifejezésből. Például:

12. LET (Z = GCFOUT(A+A**2+A**3));

Eredmény:

Z ← A $(1+A+A^2)$.

1.8.2 Behelyettesités

Kifejezésekbe való behelyettesitésre az EVAL és REPLACE rutinok szolgálnak. Ezek a rutinok a kifejezésekben előforduló meghatározott elemeket adott FORMAC kifejezésekkel helyettesitik.

Az EVAL rutin arra szolgál, hogy

- atomokat és/vagy rendszerkonstansokat FORMAC kifejezésekkel helyettesitsünk,

- függvényváltozókat és/vagy függvényeket explicit módon megadott kifejezésekkel helyettesitsünk.

A REPLACE rutin arra szolgál, hogy

- egy kifejezés meghatározott részkifejezéseit adott FORMAC kifejezésekkel helyettesitsük.

$$\text{EVAL}(\text{expr},a_1,b_1,a_2,b_2,\ldots,a_n,b_n)$$

ahol expr egy kifejezés, amelyet az argumentumok listáján az (a_i,b_i) rendezett argumentum-párok követnek. Két eset lehetséges:

- minden a_i argumentum atom, vagy FORMAC rendszerkonstans, a b_i argumentumok pedig tetszőleges FORMAC kifejezések: ekkor az a_i argumentumnak az expr kifejezésben való minden előfordulása b_i-vel helyettesitődik;

- valamelyik a_i argumentum egy m-változós függvény /pl. SIN($(1))/, vagy függvényváltozó /pl. F.($(1), $(2))/ és a neki megfelelő b_i kifejezés tartalmazza a függvény vagy függvényváltozó argumentumait: ekkor az expr kifejezésen belül az a_i-nek megfelelő függvénykifejezés minden előfordulása b_i-vel helyettesitődik. Eközben a függvény vagy függvényváltozó argumentumai helyére is behelyettesitődnek az aktuális argumentumok /l. a 2. példát/.

Mindkét esetben a behelyettesités minden a_i-re párhuzamosan /egyidejüleg/ történik.

$$\text{REPLACE}(\text{expr},a_i,b_1,a_2,b_2,\ldots,a_n,b_n)$$

ahol expr,a_i,b_1,...a_n,b_n tetszőleges FORMAC kifejezések. Az a_1 minden egyes előfordulásakor az expr kifejezésben b_1-gyel helyettesitődik. Majd az a_2 minden egyes előfordulásakor b_2-vel helyettesitődik stb. A behelyettesités itt tehát sorban és nem párhuzamosan történik. Mivel paraméterei tetszőleges kifejezések lehetnek, a REPLACE rutin az általánosabb a két behelyettesitési lehetőség közül.

Példák és megjegyzések:

1.
```
LET (X =#P*(R-RHO)**2;
     Y = EVAL(X,#P,3.141592,R,A+B));
```

Eredmény:

$Y \leftarrow 3.141592(A+B-RHO)^2$

2.
```
LET (X = SIN(A)*SIN(B+C);
     Y = EVAL(X,SIN($(1)),$(1)-$(1)**3/6));
```

Eredmény:

$Y \leftarrow (A-1/6A^3)(B+C-1/6(B+C)^3)$.

Ez a példa az EVAL alkalmazását abban a gyakori esetben mutatja be, amikor az a_i argumentum, amelynek a helyére be kell helyettesitenünk, nem szimbolikus változó, hanem függvény, példánkban SIN($(1)). A b_i kifejezés pedig, amit be kell helyettesitenünk, tartalmazza a függvény argumentumát. Ennek hatására a SIN függvény minden expr-beli előfordulásának helyére a b_i kifejezés helyettesitődik be; eközben a szimbolikusan kijelölt argumentumok helyére behelyettesitődnek az aktuális argumentumok: példánkban $(1) helyére A, illetve B+C.

3. Az EVAL rutint igen gyakran használjuk arra, hogy egy <u>függvényváltozót</u> egy FORMAC kifejezéssel helyettesitsünk:

```
LET (X = A+F.(U,V);
     Z = EVAL(X,F.($(1),$(2)),$(1)**2-2*$(1)*$(2)));
```

Eredmény:

$Z \leftarrow A+U^2-2U\ V$.

4.
```
LET (Y = 2*SIN(X)*LOG(X)*COS(X);
     Z = REPLACE(Y,2*SIN(X)*COS(X),SIN(2*X)));
```

Eredmény:

$Z \leftarrow SIN(2X)LOG(X)$.

5. Az alábbi példával illusztráljuk a különbséget az EVAL rutin <u>párhuzamos</u> és a REPLACE rutin <u>soros</u> müködése között:

```
LET (Y = A*(A+B)+C*(A+D));
```

A

```
LET (Z = EVAL(Y,A,B,B,E,C,A));
```

utasitás hatására az Y kifejezésben

A mindenütt B-vel,

B mindenütt E-vel,

C mindenütt A-val

helyettesitődik egyidejüleg.

Igy tehát

Z ← B(B+E)+A(B+D)

lesz.

Ezzel szemben a

```
LET (W = REPLACE(Y,A,B,B,E,C,A));
```

utasitás hatására az Y kifejezésben először A mindenütt B-vel helyettesitődik. Ennek eredménye:

$B(B+B)+C(B+D) \rightarrow 2B^2+C(B+D)$.

Ezután B mindenütt E-vel helyettesitődik. Igy a

$2E^2+C(E+D)$

kifejezést kapjuk. Végül C mindenütt A-val helyettesitődik. Igy a végeredmény

$W \leftarrow 2E^2+A(E+D)$

lesz.

Megjegyezzük, hogy a CHAIN müvelet segitségével a fenti példákat a valamivel egyszerübb

```
LET (X = CHAIN(A,B,B,E,C,A);
     Y = A*(A+B)+C*(A+D));
LET (Z = EVAL(Y,X));
```

illetve

```
LET (W = REPLACE(Y,X));
```

alakban is felirhatjuk.

A FORMAC73 rendszer mind az EVAL, mind pedig a REPLACE utasitás szintaxisát lényegesen kiterjesztette. Az EVAL utasitásban előforduló argumentumok jelölésére a FORMAC73 rendszer $(n) helyett az $név jelölést használja, ahol "név" egy tetszőleges azonositó. A 2. és 3. példát tehát a FORMAC73 rendszerben igy kell irnunk:

2a/
```
LET (X = SIN(A)*SIN(B+C);
     Y = EVAL(X,SIN($A),$A-$A**3/6));
```

illetve

3a/
```
LET (X = A+F.(U,V);
     Z = EVAL(X,F.($P,$Q),$P**2-2*$P*$Q));
```

Az uj jelölésmód kényelmesebb, a matematikai jelöléshez kö-

zelebb áll. A név elé helyezett $ /dollár/ jel a matematikai logika univerzális kvantorának felel meg[x]. Igy egy lépést tettünk egy olyan "matematikaibb" jelölésmód felé, amelyben pl. a 2a/ példát a

```
FOR ALL A EVAL (X,SIN(A),A-A**3/6)
```

formában képzelhetjük el. Ehhez hasonló jelölésmódot használ például a REDUCE2 nyelv [13].

Ennél lényegesen nagyobb előny azonban az, hogy az uj jelölés lehetővé teszi azt, hogy az EVAL függvényt rekurzivan hivjuk, és hogy az EVAL szerepelhessen függvénydefiniciók, azaz

```
LET(FNC...)= ...
```

alaku utasitások jobboldalán is. Az eddigi $(n) jelöléssel ez nem volt lehetséges, mert nem tudtuk a különböző függvények argumentumait egymástól megkülönböztetni.

Definiáljuk például a következőképpen az EXF függvényt, amely egy F függvényváltozóra alkalmazva, annak argumentumát kifejtett alakra hozza:

```
LET (FNC(EXF) = EVAL($(1),F.($X),F.(EXPAND($X))));
```

Itt $(1) az EXF függvény argumentumát, $X pedig az F függvényváltozó argumentumát jelöli.

A

```
LET (Z = EXF(F.((A+B)*(A-B))));
```

eredményeképpen ekkor

$$Z \leftarrow F.(A^2 - B^2)$$

lesz.

A következő példa az EVAL függvény rekurziv hivását mutatja be:

```
LET (Z = A+SIN(A+B)*F.(2*A**2);
     R = EVAL(Z,F.($X),EVAL(F.($X),A,B)));
```

Eredmény:

$$R \leftarrow A+SIN(A+B)\ F.(2\ B^2\ );$$

---

[x] Univerzális kvantornak nevezik a matematikai logikában az $\forall$ jelet /angolul: "for all..." vagy magyarul: "minden ...-re"/. Segitségével fogalmazhatjuk meg például a $\forall x(x=x)$ azaz: minden x-re igaz, hogy x=x állitást.

azaz: A helyére B helyettesitődött be minden olyan helyen, ahol A az F függvény argumentumában fordult elő.

Ezeket a példákat a régi jelölésekkel nem lehetett leirni.

A FORMAC73-ban a behelyettesités másik eszközének, a REPLACE függvénynek a szintaxisa is kibővült.

A FORMAC73 a REPLACE függvényben is megengedi a formális paraméterek, "$-változók" használatát, amit az eredeti FORMAC nyelv nem engedett meg. Leirhatunk tehát ilyen behelyettesitési szabályokat:

```
LET (EXPR = SIN(ALFA)**2+SIN(BETA)**2;
     T = REPLACE(EXPR,SIN($X)**2,1-COS($X)**2));
```

Eredmény:

T ← 2-COS(ALFA)2 - COS(BETA)2

Megjegyezzük, hogy az EVAL, illetve REPLACE függvénnyel leirt behelyettesitési szabályokban a_i-ben két "$-változó"-t nem kapcsolhat össze + vagy * jel; tehát nem fordulhat elő például $X+$Y vagy $Y*$Y.

Figyelmeztetés! A FORMAC73 rendszerben az EVAL és REPLACE behelyettesitési szabályokban függvények, függvényváltozók argumentumainak jelölésére már csak a $név alaku változókat használhatjuk, a $(n) alakuakat nem! A FORMAC73-ban tehát a 2. és 3. példát már csak a 2a/ és 3a/ szerinti formában irhatjuk. Az EXF függvény definiciójában előforduló $(1) paraméter ennek nem mond ellent: ez ugyanis itt az EXF függvény első paraméterét jelenti. A LET(FNC(...)=...) alaku függvénydefiniciókban azonban a FORMAC73-ban is $(1),$(2),... jelöli a formális argumentumokat. Ez a kis következetlenség enyhén zavaró lehet, vigyázzunk a paraméterek használatánál!

1.8.3 Differenciálás

A FORMAC nyelvben kifejezéseket szimbólikusan lehet differenciálni. Erre a

DERIV és a DRV

FORMAC rutinok és a DIFF pszeudováltozó szolgálnak.

A DERIV rutin egy kifejezésnek egy vagy több változó szerinti analitikus differenciálását végzi el.

A rutin hivása:

DERIV (expr,v_1,n_1,v_2,n_2,...,v_m,n_m)

ahol a v_1,v_2,...v_m argumentumok a kiértékelés után atomi változók, n_1,n_2,...,n_m pedig nem negativ egész értékek.

A DERIV rutin eredményül a

$$\frac{\partial^n \text{expr}(v_1,v_2,\ldots,v_m)}{\partial v_1^{n_1}\,\partial v_2^{n_2}\ldots\partial v_m^{n_m}}$$

kifejezést adja /ahol $n=\sum_{i=1}^{m} n_i$/, azaz az expr(v_1,...,v_m) kifejezés v_1 szerinti n_1-edik, v_2 szerinti n_2-edik, ... és v_m szerinti n_m-edik parciális differenciálhányadosát.

Az n_i paraméterek bármelyike elhagyható: ilyenkor a rutin alapértelmezés szerint az n_i=1 értéket feltételezi.

Az expr kifejezés tartalmazhat függvényváltozót is.

Példák:

1.
```
LET (P = 3*SIN(A*X)+A*X**2;
     Q = DERIV(P,X);
     R = DERIV(P X,2,A));
```

Eredmény:

Q ← 3A COS(A X) + 2A X

R ← $-3A^2X$ COS(A X) - 6A SIN(A X) +2

Itt Q a P kifejezés X szerinti első differenciálhányadosa. Vegyük észre, hogy a DERIV rutint csak két argumentummal hivtuk, kihasználva, hogy ha a harmadik paramétert elhagyjuk, 1 lesz az alapértelmezés szerinti értéke. R pedig a P kifejezés harmadrendü parciális deriváltja, ahol P-t az X változó szerint 2-szer, az A változó szerint pedig 1-szer differenciáltuk.

2.
```
LET (E = DERIV(G.(2*X*Y,Y),X);
     F = DERIV(G.(2*X*Y,Y),X,2));
```

Eredmény:

E ← 2YG.$^{(1)}$(2X Y,Y)

F ← $4Y^2$G.$^{(1^2)}$(2X Y,Y)

Ez a példa a DERIV rutinnak függvényváltozókra való alkalmazását illusztrálja. Figyeljük meg a jelölést:

G.$^{(1)}$(2X Y,Y) a G függvényváltozó első argumentuma szerinti <u>első</u> deriváltját jelenti:

$$\frac{\partial G}{\partial arg_1}$$

G.$^{(1^2)}$(2X Y,Y) a G függvényváltozó első argumentuma szerinti <u>második</u> deriváltját jelenti:

$$\frac{\partial^2 G}{\partial arg_1^2}$$

Mivel a G függvényváltozó specifikálatlan függvény, a differenciálást itt csak "kijelöli" a rendszer.

Vegyük észre azt is, hogy itt az X <u>atom</u> szerint differenciáltunk. Mivel a G függvényváltozó első argumentuma egy, az X atomot tartalmazó kifejezés, a DERIV rutin a közvetett függvények differenciálási szabályát alkalmazta.

Egyes számításoknál arra lehet szükség, hogy egy függvényváltozót valamelyik argumentuma szerint differenciáljunk, ahol az argumentum kifejezés is lehet. A DERIV rutin erre nem alkalmazható, mivel csak atomok szerint tud differenciálni. Erre a feladatra a DRV rutint használhatjuk.

DRV(funcvar.(arg_1,...,arg_m), \$(1),n_1,...,\$(m),n_m)

A DRV rutin első argumentuma egy függvényváltozó, arg_1,...,arg_n a függvényváltozó argumentumai. \$(i) az i-edik argumentumot jelöli, n_i pedig azt adja meg, hogy az i-edik argumentum szerint hányszor kell differenciálni. Ha n_i=1, akkor elhagyható, ugyanugy, mint a DRV rutin esetében.

Példa:

3.
```
LET(P = DRV(G.(2*X*Y,Y),$(1));
    Q = DRV(G.(2*X*Y,Y),$(1),2)) ;
```

Eredmény:

P ← G.$^{(1)}$(2X Y,Y)

Q ← G.$^{(1^2)}$(2X Y,Y)

A FORMAC nyelvben arra is van lehetőség, hogy egy függvényváltozó elsőrendü parciális deriváltjait explicit módon definiáljuk. Ezt a DIFF pszeudováltozó segitségével tehetjük meg a következő formában:

DIFF(funcvar) = CHAIN($expr_1$,$expr_2$,...,$expr_n$)

Itt a DIFF pszeudováltozó argumentuma, a "funcvar" függvényváltozó, egy n-változós függvényt jelöl. A CHAIN müvelet argumentumaként megadott $expr_1$,$expr_2$,...,$expr_n$ kifejezések a "funcvar" függvényváltozó első, második, ..., n-edik argumentuma szerinti differenciálhányadosát definiálják.

A DIFF pszeudováltozót azért nevezzük igy, mert formailag egy értékadó utasitás baloldalán szerepel, ahol a FORMAC nyelv szintaxisa szerint csak változó állhat. A fenti utasitás eredménye az, hogy az F függvényváltozóhoz hozzárendelődik parciális deriváltjainak sorozata.

Példa:

4.
```
LET (Y = F.(3*X);
    DIFF(F) = CHAIN($(1)**2+A**2));
```

F itt egy egyváltozós függvény. A DIFF utasitással az első /és egyetlen/ argumentuma szerinti első differenciálhányadosát a következőképpen definiáltuk /matematikai jelölésmódra áttérve/:

$$\frac{\partial F}{\partial arg_1} = arg_1^2 + A^2 \quad .$$

Ha ezután az Y kifejezésre alkalmazzuk a DERIV függvényt,

```
LET (R = DERIV(Y,X));
```

az eredmény

$$R \leftarrow 3(9X^2+A^2)$$

Ugyanakkor a

```
LET (Q = DRV(Y,$(1)));
```

utasitás eredményeképpen

$$Q \leftarrow 9X^2+A^2$$

lesz.

Vegyük észre ismét, hogy a DERIV és a DRV rutinok hatása különböző. A DERIV, ha specifikálatlan függvényre /függvényváltozóra/ alkalmazzuk, a rutin hivásakor megadott atomi változó /példánkban X/ szerinti differenciálhányadost adja, mig a DRV az F függvény argumentuma /példánkban 3*X/ szerinti differenciálhányadost.

1.8.4 Összehasonlitás

Két FORMAC kifejezést az IDENT nevü függvény segitségével hasonlithatunk össze. Az IDENT PL/I függvény értéke egy-egy bit hosszuságu PL/I bitfüzér, amelynek értéke 1, ha a két kifejezés azonos és 0 különben.

Például:

```
LET (X = (A+B)**2;
     Y = A**2+2*A*B+B**2);
IF IDENT(X;Y) THEN LET (C = E+B);
```

Bár X és Y algebrailag ekvivalensek, de nem azonosak, és ezért az IDENT (X;Y) értéke 0. Figyeljünk fel azonban arra, hogy az IDENT(EXPAND(X);Y) értéke 1.

1.8.5 Kifejezéseket részekre bontó rutinok

Az itt leirandó rutinok segitségével az egyes kifejezéseket alkotórészeikre bonthatjuk, illetve szerkezetüket vizsgálhatjuk.

A COEFF, HIGHPOW és LOWPOW rutinok egy kifejezésben szereplő valamely részkifejezés együtthatóját, legnagyobb és legkisebb hatványkitevőjét szolgáltatják.

A NUM és DENOM rutinok egy törtkifejezés számlálóját, illetve nevezőjét adják meg.

A LOP, NARGS és ARG rutinok a kifejezés szerkezetéről és argumentumairól adnak információkat.

COEFF($expr_1$,$expr_2$) - megadja az $expr_2$ együtthatóját az $expr_1$-ben.

HIGHPOW($expr_1$,$expr_2$) - megadja az $expr_2$ előforduló legnagyobb hatványkitevőjét az $expr_1$-ben.

LOWPOW($expr_1$,$expr_2$) - megadja az $expr_2$ előforduló legkisebb hatványkitevőjét az $expr_1$-ben.

NUM(expr) - az expr törtkifejezés számlálóját adja, vagy pedig magát az expr értékét, ha expr nem törtkifejezés.

DENOM(expr) - az expr törtkifejezés nevezőjét adja, vagy az 1 értéket adja, ha az expr kifejezés nem tört.

LOP(var)
NARGS(var)
ARG(n,expr)
} - a LOP, NARGS és ARG rutinok a var kifejezés szerkezetéről szolgáltatnak információt.

A rutinok részletesebb ismertetése előtt néhány definiciót előre kell bocsátanunk.

A kifejezéseket aszerint, hogy milyen müveleteket tartalmaznak, és azokat milyen sorrendben kell elvégezni, különböző tipusokba soroljuk. A kifejezés lehet

- összegkifejezés, pl. 3*A*B+C;
- szorzatkifejezés, pl. 3*A*(B+C);
- hatványkifejezés, pl. A**3;
- függvénykifejezés, pl. SIN(A)

stb.

A kifejezésben szereplő fő müvelet az a müvelet, amely a kifejezés tipusát meghatározza. Példáink közül

- a 3*A*B+C kifejezésben a fő müvelet az összeadás /+/;
- a 3*A*(B+C) kifejezésben a fő müvelet a szorzás /*/;
- az A**3 kifejezésben a fő müvelet a hatványozás /**/;
- a SIN(A) kifejezésben a fő müvelet a SIN függvény.

A különböző függvények operandusainak /argumentumainak/ száma különböző:

- a hatványozásnak két operandusa van, az alap és a kitevő: az A**3 kifejezésben A és 3;
- a függvénykifejezéseknek a szereplő függvénytől függően egy vagy több argumentuma van: pl. a SIN(A) kifejezésnek egy argumentuma van, A. A COMB(N,K) kifejezésnek két argumentuma van, N és K;
- az összeadásnak és szorzásnak két vagy több argumentuma lelet. Például az A*B kifejezésben a szorzásnak két argumentuma van, A és B; a 3*A*(B+C) kifejezésben a szorzásnak három argumentuma van, 3, A és (B+C).

Ha egy kifejezés szerkezetét elemezni akarjuk, a következő kérdéseket tehetjük fel:

a/ Mi a fő müvelet a kifejezésben?
Például a 3*A*(B+C) kifejezésben a szorzás /*/.

b/ Hány argumentuma /operandusa/ van a fő müveletnek?
Például a 3*A*(B+C) kifejezésben a szorzásnak három argumentuma van: 3, A és (B+C).

c/ Mi a kifejezésben a fő müvelet i-edik argumentuma /operandusa/?
Például a 3*A*(B+C) kifejezés harmadik argumentuma /háromtényezős szorzat harmadik tényezője/ (B+C).

Ezekre a kérdésekre adnak választ a LOP, NARGS, ARG rutinok.
A

LOP(var)

függvény értéke egy FIXED(31,0) tipusu PL/I egész szám, amely a "var" kifejezés fő müveleti jelének megfelelő egész kódot adja. A kódokat [1] 36. oldalán megtalálhatjuk, itt csak a néhány legfontosabbat közöljük:

A LOP rutin által szolgáltatott legfontosabb müveleti kódok:

Müvelet	Kód /LOP értéke/
CHAIN	11
DERIV	12
PLUS /+/	24
MINUS /-/	25
TIMES /*/	26
EXPON /**/	31

Az

NARGS(var)

függvény értéke egy FIXED(31,0) tipusu PL/I egész szám, amely a "var" kifejezésben szereplő fő müveleti jel argumentumainak számát adja meg.

Figyeljünk fel arra, hogy a LOP(var) és a NARGS(var) függvény értéke egy PL/I érték, ezért a LOP és NARGS rutinokat csak PL/I utasitásokban hivhatjuk, FORMAC utasitásokban nem!

Az

ARG(n,expr)

függvény argumentumai: n egy nem-negativ FORMAC egész szám, expr pedig egy kifejezés.

Ha n=0, ARG értéke az expr kifejezéssel egyenlő.

Ha n>0 és expr fő müveletének legalább n operandusa van, ARG értéke az n-edik operandus lesz; különben ARG=0.

Igy, ha expr egy n-tagu összeg, ARG(1,expr) az első tagot, ha expr egy n-tényezős szorzat, ARG(1,expr) az első tényezőt adja.

Példák:

1.
```
LET (Y = A*X**3+2*A*X;
     C1 = COEFF(Y,X**3);
     C2 = COEFF(Y,X));
```
Eredmények:
```
C1 ← A;
C2 ← 2A.
```

2.
```
LET (P = 3*X**2+4*X+5;
     Q = A*G.(A+B,C));
LOPP = LOP(P);
LOPQ = LOP(Q);
```
Eredmények:
```
LOPP ← 24   /a '+' müveleti jelnek megfelelő kód/
LOPQ ← 26   /a '*' müveleti jelnek megfelelő kód/
```

3.
```
NP = NARGS(P);
NQ = NARGS(Q);
```
ahol P és Q a 2. példában szereplő kifejezések.

Eredmények:
```
NP ← 3;
NQ ← 2.
```
/P 3-tagu összeg, Q 2-tényezős szorzat/.

4.
```
LET (AP = ARG(1,P);
     AQ = ARG(1,Q));
```
ahol P és Q ismét a 2. példában szereplő kifejezések.

Eredmények:
```
AP ← 3X**2      /a P 3-tagu összeg első tagja/
AQ ← A          /a Q 2-tényezős szorzat első tényezője/
```

5.
```
LET (D = DENOM(A**2/B**2+A/B));
```
Eredmény:
```
D ← B**2.
```

A FORMAC rendszer a DENOM rutin hivása előtt automatikusan közös nevezőre hozza a kifejezést: DENOM argumentuma (A**2+A*B)/B**2 lesz!

6.
```
LET (NEVEZO = DENOM(3/4));
```
Eredmény:
```
NEVEZO ← 1.
```

Ennek az eredménynek az a magyarázata, hogy a NUM és DENOM rutin csak racionális tört kifejezések számlálóját, illetve nevezőjét tudja leválasztani. A FORMAC rendszer a racionális törtszámokat szétválaszthatatlan egységként kezeli.

1.8.6 A memória használata, felszabaditása

A memóriával való gazdálkodást, a memória felszabaditását két FORMAC rutin szolgálja.

A SAVE rutin segitségével háttértárba irhatunk egyes, ideiglenesen nem használt FORMAC kifejezéseket. Az általuk használt memóriaterület /a kifejezésekre való ujabb hivatkozásig/ felszabadul.

Az ATOMIZE rutin segitségével egy kijelölt /kifejezést tartalmazó/ FORMAC változót atomivá tehetünk: a kifejezés által elfoglalt memóriaterület felszabadul.

SAVE(var_1;var_2;...;var_n)

a var_1,var_2,...,var_n nevü FORMAC kifejezéseket a főmemóriából a háttértárba átmásolja és felszabaditja az általuk eddig elfoglalt memóriaterületet. Ha ezekre a kifejezésekre a program végrehajtása során később hivatkozunk, akkor automatikusan visszatöltődnek a memóriába. Később ujra el lehet tárolni őket.

Példa:

```
 i/    SAVE(A);
         :
ii/    LET(B=A+C);
iii/   SAVE(A;B);
         :
iv/    LET(A=X*Y);
```

Az egyes utasitások eredménye:

i/ "A" a háttértárba másolódik;

ii/ "A" visszakerül a főmemóriába;

iii/ "A" és "B" a háttértárba másolódik;

iv/ "A" megkapja az XY értékét, és "A" korábbi értéke a háttértárban törlődik.

ATOMIZE(var_1;var_2;...;var_n)

a var_1,var_2,...,var_n FORMAC változók atomi változókká válnak. Az a hely /memóriaterület/, amelyet korábban az általuk tartalmazott kifejezések foglaltak le, felszabadul.

Példa:

```
LET (X = (A+B)*(C+SIN(P))*(D+COS(Q)));
PRINT_OUT(X);
ATOMIZE(X);
```

Az X változó atomi változó lesz, a benne tárolt kifejezés által elfoglalt memóriaterület felszabadul.

Figyeljünk az irásmódra! A SAVE és az ATOMIZE rutinok argumentumait pontosvesszővel /;/ választjuk el egymástól.

1.9 OPTSET lehetőségek. A FORMAC_OPTIONS utasitás.

Az OPTSET utasitással a FORMAC program üzemmódjait változtathatjuk meg: részben az AUTSIM rutin müködését vezérelhetjük bizonyos mértékig, részben pedig a nyomtatás módjára, illetve nyomkövetésre adhatunk utasitást.

Az OPTSET lehetőségei a következők: az alapértelmezést aláhuzással jelöltük.

a/ TRANS NOTRANS
b/ INT NOINT
c/ EXPND NOEXPND
d/ EDIT NOEDIT
e/ PROPER IMPROPER
f/ LINELENGTH=m /120/
g/ TRUNC=n

Az opciók első csoportja az AUTSIM rutin müködését irányitja.

a/ TRANS opció: a FAC, COMB és STEP kivételével minden olyan transzcendens függvény /pl. SIN, LOG, ERF stb/ értékét kiszámitja, melyeknek argumentuma egy numerikus konstans. Például SIN(#P/6)→1/2, ha OPTSET(TRANS) érvényes.

b/ Az INT opcióval a FORMAC egész függvények /FAC, COMB és STEP/ értékét a rendszer kiszámitja, ha ezeknek a függvényeknek az argumentuma egy numerikus konstans. Például FAC(3)→6, ha

OPTSET(INT) érvényes.

c/ Az EXPND opcióval a FORMAC rendszer a kifejezéseket kifejtett alakban állitja elő, az asszociativ és a disztributiv szabályt használva.

Az opciók következő csoportja a program eredményeinek kiiratási formátumát vezérli.

d/ Az EDIT opció a hagyományos matematikai jelöléseknek, müveleti jeleknek megfelelő output formát eredményez.

e/ PROPER opcióval minden racionális számot az a+b/c /valódi tört/ formában lehet kinyomtatni /pl. 7/6 → 1+1/6/.

f/ LINELENGTH=m hatására a kinyomtatandó FORMAC output sorainak hossza legfeljebb m karakter lesz /az alapértelmezés 120 karakter/.

g/ Végül a TRUNC=n opció, melyet csak a FORMAC73 rendszer ismer, hatványsorokkal való müveletek végzését segiti.

A TRUNC=n opció hatására, ahol n tetszőleges pozitiv egész szám, a FORMAC rendszer egy kifejezésben mindazokat a tagokat törli, amelyeknek kitevője ≥n. Ez az opció /amelyet a FORMAC73 rendszerben vezetett be K. Bahr/ hatványsorok csonkitására használható.

Egy OPTSET utasitásban több opciót is előirhatunk, például:

```
OPTSET(EXPND;PROPER;LINELENGTH=80);
```

Figyeljünk itt is az irásmódra! Az OPTSET utasitásban megadott opciókat pontosvesszővel /;/ választjuk el egymástól.

Egy OPTSET utasitás a következő OPTSET utasitásig, vagy a program futásának végéig érvényes.

Ezenkivül létezik még egy további opció-pár is: a PRINT és NOPRINT /alapértelmezés: NOPRINT/.

OPTSET(PRINT) hatására minden értékadó LET utasitás eredménye kinyomtatódik. Ez az opció tehát a programbelövést, nyomkövetést segiti.

Az OPTSET(PRINT), illetve OPTSET(NOPRINT) utasitások müködése - az azonos név ellenére - lényegében különbözik az eddig ismertetett OPTSET utasitásétól. Az OPTSET(PRINT) utasitást ugyanis a preprocesszor dolgozza fel, és nem a program futásakor hajtódik végre. Az OPTSET(PRINT) tulajdonképpen a preprocesszor üzemmódját módositja: hatására minden LET utasitás ugy forditódik le, mintha PRINT_OUT utasitás lenne. Ezért:

- a PRINT, illetve NOPRINT opciót nem szabad más OPTSET opcióval együtt megadni, hanem csak külön OPTSET utasitásban;

- az OPTSET(PRINT) hatása a következő OPTSET(NOPRINT) utasitásig tart /és viszont/. A "következőt" azonban itt a forrásnyelvi utasitások fizikai sorrendje szerint kell érteni, és nem a program végrehajtásának dinamikus sorrendjében, mint az előzőleg ismertetett OPTSET paramétereknél.

Minden FORMAC programban kötelezően szerepelnie kell a

FORMAC_OPTIONS;

utasitásnak. Ennek az utasitásnak minden más FORMAC utasitást meg kell előznie. Ez nem végrehajtható utasitás. Szerepe az, hogy az előfeldolgozó program számára jelezze: a beolvasott forrásprogram egy FORMAC program. Hatására az előfeldolgozó program a fordítás eredményeképpen keletkező PL/I programba beépiti a szükséges deklarációkat. Ha a FORMAC programban nem szerepel a FORMAC_OPTIONS utasitás, az előfeldolgozás /látszólag/ hibátlanul érhet véget. A következő lépésben azonban, a keletkezett PL/I program forditásakor a PL/I forditóprogram hibásnak jelez minden egyes olyan PL/I utasitást, amely egy FORMAC utasitás leforditásának eredményeként jött létre. Hiányoznak ugyanis az utasitások leforditásához szükséges deklarációk.

Nem szabad elfelejtenünk tehát, hogy minden FORMAC program elején a

FORMAC_OPTIONS;

utasitásnak kötelezően szerepelnie kell.

2. A FORMAC és PL/I kapcsolata

Az első fejezetben megismertük a FORMAC nyelv utasitásait, a FORMAC függvényeket, FORMAC rutinokat s i.t. Ezek ismeretében egyszerübb FORMAC programokat már össze tudunk állitani.

Bonyolultabb FORMAC programok irásához szükségünk van arra, hogy a PL/I nyelv lehetőségeit teljesen ki tudjuk használni. Igy szükség van arra, hogy PL/I változók értékét egy FORMAC utasitásban felhasználhassuk és megforditva: FORMAC változók értékét használhassuk PL/I utasitásokban. A FORMAC nyelvnek, mint szimbolikus nyelvnek ereje többek között éppen abban van, hogy a PL/I nyelv numerikus lehetőségeit is ki tudja aknázni.

A következőkben a FORMAC és PL/I közötti kapcsolattal foglalkozunk. Ismertetjük, hogy hogyan lehet PL/I változók értékét használni FORMAC utasitásokban, illetve FORMAC változók értékét PL/I utasitásokban, hogyan kell eljárásokat irni és hivni.

2.1 PL/I változók használata FORMAC utasitásokban

Ha egy PL/I változó neve idézőjelek közé foglalva szerepel egy FORMAC utasitásban /LET, PRINT_OUT, SAVE stb/, akkor a PL/I változó értéke /amely lehet numerikus érték vagy karaktersorozat is/ helyettesitődik be a megfelelő helyre.

Például, ha

A=3 és B='2*SIN(X)'

/A és B PL/I változók, B értéke egy karaktersorozat, mégpedig egy szabályos FORMAC-kifejezés karakter képe/, akkor a

```
LET (Y = "B"+"A");
```

utasitás eredményeképpen az Y FORMAC változó értéke 2*SIN(X)+3 lesz.

A következő példa azt mutatja be, hogy hogyan használjuk ezt a lehetőséget ciklus szervezésére:

```
DO I=1 TO N;
   LET (I="I");
   ...
   LET(A(I)=B(3*I)+I);
   ...
END;
```

Itt a LET(I="I"); utasitás baloldalán az I nevü FORMAC változó szerepel. A jobboldalon az I PL/I változó nevét adtuk meg idézőjelek között. Ennek hatására az I PL/I változó értéke bekerül a baloldalt megadott FORMAC változóba. Ezután az I FORMAC változót használhatjuk további utasitásokban is /például, mint esetünkben is, indexelésre/.

Természetesen a példában szereplő FORMAC és PL/I változónak adhattunk volna különböző nevet is.

Ugyanezt a példát a következőképpen is irhattuk volna:

```
DO I=1 TO N;
   ...
   LET(A("I")=B(3*"I")+ "I");
   ...
END;
```

Ezek a példák arra mutattak lehetőséget, hogy egy PL/I változó numerikus értékét hogyan használhatjuk fel FORMAC utasitásokban.

Karakter tipusu PL/I változókban FORMAC változó nevét, vagy egy FORMAC kifejezés karakter képét helyezhetjük el. Ha ezután a PL/I változó neve egy FORMAC utasitásban idézőjelek közé téve szerepel, a megfelelő helyre a PL/I változó értéke, azaz a FORMAC változó vagy kifejezés helyettesitődik be. Az igy létrejövő utasitásnak egy szintaktikusan helyes FORMAC utasitásnak kell lennie.

Például legyen A egy karakter tipusu PL/I változó, X, Y és Z pedig FORMAC változók. Az

```
A = 'X';
LET("A"=Y**2+Z**2-1);
```

utasitások hatására az X FORMAC változó értéke Y^2+Z^2-1 lesz.

2.2 FORMAC változók PL/I eljárások paramétereként

Az előző pontban leirt, bonyolultnak tünő mechanizmust arra használhatjuk, hogy PL/I eljárások paramétereként FORMAC változókat, illetve kifejezéseket is megadhassunk.

Erre egy példát mutatunk be:

```
PELDA: PROCEDURE (A,B);
       DECLARE (A,B) CHAR(8);
       LET("B"="A"+3);
     END PELDA;
     ...
     CALL PELDA ('X+2','BETA');
     ...
```

A PELDA eljárás fenti hivásának hatására a

```
LET (BETA = X+2+3);
```

FORMAC utasitás hajtódik végre és a BETA FORMAC változóba az X+5 kifejezés kerül.

Ezután megfogalmazhatjuk azt a szabályt, hogy hogyan adhatunk meg FORMAC változókat és kifejezéseket PL/I eljárások paramétereként.

A PL/I eljárás deklarációjában azokat a paramétereket, amelyekbe a híváskor FORMAC változót vagy kifejezést akarunk helyettesiteni, karakter-tipusu változóként kell deklarálni. Ha az eljáráson belül ezeket a paramétereket LET, PRINT_OUT, SAVE utasitásokban használjuk, nevüket tegyük idézőjelbe /a példában A és B volt ilyen paraméter/.

Az eljárás hivásakor aktuális paraméterként egy FORMAC változót vagy kifejezést adunk meg aposztrofok között /mint az előző példában is/; formailag tehát a paramétereket PL/I karakterkonstansként adjuk meg.

Egy másik lehetőség: az aktuális paraméter lehet PL/I karakterváltozó is, amelybe előzőleg beirtunk egy FORMAC változónevet vagy egy kifejezést. Az előző pont példáját igy is módosithatjuk:

```
PELDA: PROCEDURE(A,B);
       ...
       END PELDA;
       ...
       DECLARE(P,Q) CHAR(8);
       P = 'X+2';
       Q = 'BETA';
       CALL PELDA (P,Q);
```

A FORMAC változók paraméterként való használatáról azt mondhatjuk, hogy a hivás itt <u>név szerint</u> és nem érték szerint történik /az eljáráshivásban a kifejezés vagy változó <u>neve</u> szerepel és nem

egy mutató/. Ez azért lehetséges, mert a FORMAC változók hatásköre univerzális, az egész programra kiterjed.

Eljáráshivásnál a programozó felelős azért, hogy az eljárás törzsében, vagy aktuális paraméterként használt karakterváltozók hossza elegendő legyen a megfelelő FORMAC változó, illetve kifejezés elhelyezésére. Például ha a fenti példát igy módositjuk:

```
DECLARE(P,Q) CHAR(8);
P = 'ALFA+BETA**2+GAMMA**3';
Q = 'BETA';
CALL PELDA (P,Q);
```

a program hibás lesz, mert a P változó hossza csak 8 karakter.

Ugyancsak a programozónak kell ügyelnie arra, hogy mely paraméterek értéke lehet csak egy FORMAC változó neve, és mely paraméter lehet kifejezés is. Például, ha PELDA a 2.1 pontban definiált eljárás, a

```
CALL PELDA ('BETA','X+2');
```

eljáráshivás hibás, mert itt az eljárás mádosik paramétere, mivel értékadó utasitás baloldalán szerepel, csak változónév lehetne. Igy a

```
LET (X+2=BETA+3);
```

szintaktikusan hibás FORMAC utasitás keletkezik.

2.3 A CHAREX és a CONVERT utasitás

Ezek az utasitások is a PL/I és FORMAC változók közötti értékadással kapcsolatosak.

A CHAREX utasitással egy PL/I változóban karaktersorozatot helyezhetünk el. Formája:

```
CHAREX (var=változó),
```

ahol "var" egy PL/I változó hosszuságu VARYING karaktersorozat, A "var" karakterváltozóhoz a 'változó=kifejezés' karaktersorozatot rendeli hozzá, ahol "változó" az utasitás jobboldalán szereplő FORMAC változó, "kifejezés" pedig a változó aktuális értéke.

Példa:

```
DECLARE X CHARACTER(80) VARYING;
LET(Y=A/B; Z=SIN(3*Y*B)/B);
CHAREX (X=Z);
```

eredménye

X ← 'Z=B**(-1)+SIN(3*A)'

Megjegyzendő, hogy a CHAREX által előállitott karaktersorozat itt egy szabályos PL/I utasitás /eltekintve attól, hogy a bezáró pontosvessző (;) hiányzik/.

Igy a programozó egy FORMAC programban elő tud állitani PL/I utasitásokat, és ezeket beilleszthetí egy PL/I programba. Igy pl. egy FORMAC kifejezés számértékét határozhatjuk meg.

A CONVERT utasitás formája:

CONVERT FIXED(x_1;...;x_n)FLOAT(y_1;...;y_m);

ahol x_1,...,x_n és y_1,...,y_m index nélküli PL/I változók, és a FIXED, illetve FLOAT listák egyike hiányozhat. A CONVERT utasitásnak mindig a program elején kell állnia, a FORMAC_OPTIONS utasitás után, de a végrehajtható utasitásokat megelőzve.

Ha ezen változók valamelyike egy FORMAC értékadó utasitás baloldalán idézőjelek között szerepel, akkor a jobboldalon szereplő kifejezés értéke /ez csak számérték lehet/ bekerül a baloldalon álló PL/I változóba.

Ha pedig egy, a CONVERT utasitásban felsorolt PL/I változó egy FORMAC értékadó utasitás jobboldalán szerepel idézőjelek között /vagy értékadó utasitás baloldalán, de indexként/, akkor a rendszer egy vele azonos nevü FORMAC változót hoz létre. Ebbe a FORMAC változóba bekerül a PL/I változó értéke, a CONVERT utasitás által kijelölt tipusra való konvertálás után.

Ha a változó a FIXED listán szerepel, akkor FIXED BINARY (31,0) alakra konvertálódik, ha pedig a FLOAT listán, akkor FLOAT BINARY(53) alakra.

Megjegyzés: A CONVERT utasitás tehát nem a változó tipusát definiálja, hanem azt, hogy milyen konverzió hajtódik végre, ha a változó idézőjelek között szerepel egy FORMAC utasitás jobboldalán.

Példa:

```
CONVERT FLOAT(A);
   :
LET ("A"=X+Y);
```

/ahol A egy PL/I változó/ hatása ugyanaz, mint a

```
LET (A=X+Y;
     A=ARITH(A));
```

utasitásoknak /az ARITH függvény ismertetését lásd a következő pontban/.

2.4 FORMAC számkonstansok használata PL/I utasitásokban

Az INTEGER és az ARITH függvény segitségével alakithatunk át FORMAC számkonstansokat PL/I számkonstansokká.

A függvények hivásának formája:

```
INTEGER(var)
```

illetve

```
ARITH(var)
```

ahol "var" egy olyan FORMAC változó, amelynek aktuális értéke egy FORMAC számkonstans.

Az INTEGER függvény ezt az értéket egy BINARY FIXED (31,0) tipusu PL/I változóvá konvertálja /a lebegőpontos, illetve racionális értékeket kerekiti/. Az ARITH függvény a számértéket kétszeres hosszuságu BINARY FLOAT(53) tipusu lebegőpontos számmá konvertálja.

Példa:

Ha az A, B, C FORMAC változók értéke rendre 4, 5.2, illetve 11/4, X és Y pedig PL/I változók, akkor az

```
X = INTEGER(A)+INTEGER(B)+INTEGER(C);
Y = ARITH(A)+ARITH(B)+ARITH(C);
```

PL/I utasitások eredménye rendre X ← 12, illetve Y ← 1.195 E01 lesz. /Az INTEGER függvény a számokat a legközelebbi egész értékre kerekiti./

2.5 FORMAC eljárások

A 2. fejezetben eddig elmondottakból láthatjuk, hogy a PL/I nyelv eljáráshivási mechanizmusát FORMAC programokban eléggé nehézkes használni. FORMAC változókat PL/I eljárások paramétereként átadni csak bonyolult és kényelmetlen módon lehet.

A FORMAC73 rendszer ezért bevezette a FORMAC eljárás /procedure/ fogalmát, amelynek paraméterei FORMAC változók, illetve

kifejezések lehetnek. A paraméterátadás "érték szerint" történik.

A FORMAC eljárás kezdetét az F_PROCEDURE kulcsszó, végét F_END jelöli. Az eljárás értéket az F_RETURN(var) utasitáson keresztül adhat vissza a hivó programnak.

Ha az F_RETURN kulcsszó után nem adunk meg változónevet, akkor a hivó program az eljárás értékeként a DENARGS rendszerváltozó értékét kapja meg. A programozó a DENARGS rendszerváltozónak értéket adhat az eljárásban. Ezt egy példával az 5.2 pontban mutatjuk be.

Az eljárást a hivó programban az F_FUNCTION kulcsszóval deklarálni kell.

Az eljáráson belül LOCAL kulcsszóval lokális FORMAC változókat deklarálhatunk, ezekenek kezdőértéket is adhatunk. Ha nem adunk a lokális változónak kezdőértéket, akkor alapértelmezés szerint 0-val lesz egyenlő.

Példa FORMAC eljárásra:

```
SUM: F_PROCEDURE(EXPR,IND,A,B)
     LOCAL(S); /*S=0 ALAPERTELMEZES SZERINT*/
          DO I = INTEGER(A) TO INTEGER(B);
          LET(S = S+EVAL(EXPR,IND,"I"));
     END;
     F_RETURN(S);
F_END SUM;
```

Példák az eljárás hivására:

a/ e=2.718281828... közelitő értékének kiszámitása:

```
F_FUNCTION(SUM);
...
LET (R=1.0*SUM(1/FAC(I),I,0,20));
```

Az 1.0-val való szorzás azért szükséges, hogy az eredményt lebegőpontos alakban kapjuk meg racionális tört helyett.

b/ sin(x) Taylor-sorfejtésének előállitása:

```
F_FUNCTION(SUM);
...
LET(S=SUM((-1)**(M-1)*X**(2*M-1)/FAC(2*M-1),M,1,10));
```

Az OPTSET utasitás FORMAC eljáráson belül is használható. Ügyeljünk azonban arra, hogy az OPTSET utasitás hatása globális, nem válik érvénytelenné az eljárásból való visszatéréskor. Erről a programozónak kell gondoskodnia a visszatérés előtt, ha szüksé-

ges, például az alábbi módon:

```
FORMFUNC: F_PROC(X,Y)
          LOCAL (P = A+B; Q = A-B; R);
               OPTSET (EXPND);
               ...    az eljárás törzse
               OPTSET (NOEXPND);
          F_RETURN (R);
F_END FORMFUNC;
```

A példa szemlélteti az OPTSET használatát FORMAC eljáráson belül, és azt, hogy lokális változó kezdőértéke kifejezés is lehet. Vegyük észre azt is, hogy az F_PROCEDURE kulcsszó a PROCEDURE kulcsszóhoz hasonlóan rövidithető.

FORMAC eljárásként definiált függvény szerepelhet függvénydefiniciókban is.

3. FORMAC feltételek

A PL/I-ben használható ON CONDITION feltételeken kivül a FORMAC programokban öt további ON CONDITION feltétel is használható. Ezek segitségével a programozó a FORMAC utasitások végrehajtásakor fellépő hibák esetén a programnak egy hibajelző, hibafeldolgozó részére adhatja át a vezérlést.

A FORMAC feltételek:

1. DENSYN - Syntactic Errors /szintaktikus hibák/

Példák:

A DENSYN feltétel következik be az alábbi FORMAC utasitások végrehajtásakor:

```
LET (Y=A/(B*SIN(C,B)));
    mert a SIN függvénynek csak egy argumentuma lehet;
LET (A=X**-3);
    mert a - /minusz/ jel nem állhat közvetlenül más
    müveleti jel után, az utasitás helyesen
    LET(A=X**(-3)); lenne.
```

2. DENSEM - Semantic Errors /szemantikus hibák/

Példa:

A DENSEM feltétel következik be, ha a

```
LET (Y = DERIV(F,X,N));
```

utasitás végrehajtásakor az N FORMAC változó értéke nem egy nem-negativ egész szám, például N=-2.5.

3. DENSYSM - System Errors /rendszerhibák/

Ennek a feltételnek elvileg nem volna szabad bekövetkeznie. Azt jelenti, hogy a program olyan hibát tartalmaz, amelyet a FORMAC rendszer nem tudott megfelelően kielemezni. A felhasználó az ilyen hibákat programjában esetleg csak többszöri próbálkozással tudja behatárolni.

4. DENSIZE - Size Errors

A hiba jelentése: a program rendelkezésére álló memória kimerült. A programozó ujra futtathatja a programot ugy, hogy nagyobb memóriaterületet /REGION/ jelöl ki: vagy pedig célszerü megnézni, hogy a SAVE, ATOMIZE utasitásokkal lehet-e memóriaterületet felszabaditani.

5. DENERRR

Az eddig leirt feltételek bármelyikének bekövetkezésekor a DENERRR feltétel is bekövetkezik.

Példa az ON feltételek használatára:

ON CONDITION (DENSYN) GOTO L2;

ahol L2 egy utasitás cimkéje, ahol a felhasználó a hiba feldolgozása, esetleges hibajelzés kiirása után folytathatja a programot. Igy az esetleges szintaktikusan hibás utasitás nem okozza a futtatás befejezését, hanem utána még folytatódhat a további utasitások szintaktikus elemzése és végrehajtása.

Az

ON CONDITION (DENERRR) GOTO L;

utasitás hatására bármelyik hibafeltétel /DENSYN, DENSYM, DENSIZE, DENSYSM/ bekövetkezése esetén az L cimkére adódik a vezérlés.

Az ON CONDITION utasitás a forrásprogramban bárhol előfordulhat.

4. FORMAC program végrehajtása

Egy teljes FORMAC job végrehajtása logikailag négy lépésből áll:

FORMAC makró processzor futtatása
PL/I fordítás
Szerkesztés /Linkage Editor/
Végrehajtás

Ennek megfelelően a FORMAC jobok futtatására összeállitott Job Control eljárás is négy job lépésből áll.

4.1 A FORMAC makró preprocesszor

A FORMAC makró preprocesszor, a MINIMAC, egy load modul-könyvtárból töltendő be és futtatandó. Paraméterei nincsenek.

A MINIMAC makróprocesszor a FORMAC forrásnyelvü programot a SYSIN input periférián olvassa be.

A FORMAC forrásprogram egy PL/I program, amely FORMAC utasitásokat is tartalmaz. Ezért kötelezően a

```
programnév: PROCEDURE  OPTIONS(MAIN);
```

utasitással kezdődik.

A programban kötelezően szerepelnie kell a

```
FORMAC_OPTIONS;
```

utasitásnak, mely a preprocesszor számára jelzi, hogy FORMAC programról van szó. Ez az utasitás minden más FORMAC utasitást megelőz, a végrehajtható utasitásokat is, valamint az OPTSET és CONVERT utasitásokat /ha ezek szerepelnek/.

Az utasitások kötelező sorrendjét a 4.6 pontban, illetve az 5. fejezet mintaprogramjaiban mutatjuk be részletesen.

A FORMAC programot - a PL/I szabályai szerint - az

```
END;
```

utasitás fejezi be.

A MINIMAC program egy makróprocesszor, amely a beolvasott FORMAC utasitásokat PL/I utasitásokra /a legtöbb esetben szubrutinhivásokra/ forditja le. A program tartalmazza a forditáshoz szükséges makródefiniciós táblázatot.

A feldolgozás eredményeképpen létrejött PL/I programot a pre-

processzor a SYSUT3 periférián irja ki. A SYSPRINT periférián listázza a forrásprogramot és az esetleges hibajelzéseket.

Itt fel kell hivnunk a figyelmet arra, hogy a MINIMAC program nem ismeri a tárgynyelv, esetünkben a PL/I nyelv szintaxisát és szemantikáját. Igy nem vizsgálja azt, hogy a generált PL/I program nem tartalmaz-e hibákat. Ezért gyakran előfordul, hogy a FORMAC program irásakor elkövetett hibák következtében csak a PL/I forditásban kapunk hibajelzést. Például egy gyakori hiba, ami ezt eredményezheti: egy LET utasitás bezáró zárójele hiányzik. Ilyenkor általában elég nehéz a PL/I forditás listája alapján visszakeresni az eredeti FORMAC programban a hibás utasitást.

4.2 PL/I forditás

A MINIMAC preprocesszor által készitett PL/I programot a PL/I forditóprogram forditja le a szokásos módon. Ügyeljünk azonban arra, hogy a MINIMAC program output-ja a kártyának mind a 80 oszlopát használja, ezért a PL/I forditóprogram sorszélesség-opcióját /SORMGIN paraméter/ ennek megfelelően kell beállitani, azaz a PARM='SORMGIN=(1,80)' paramétert kell megadni. Bármilyen, általában hozzáférhető PL/I lehetőség felhasználható, például a PL/I hibakeresési /debug/ szolgáltatásai.

4.3 Szerkesztés

A FORMAC job szerkesztési lépése ugyanaz, mint egy PL/I job szerkesztési lépése. Azt a könyvtárat, amely a FORMAC program futtatásakor használt load modulokat tartalmazza, láncolni kell a rendszer PL/I load könyvtárához /SYS1.PL1LIB/ a JCL szabályai szerint.

4.4 Futtatás

A SYSIN, SYSPRINT és más adatállományokhoz szükséges DD utasitásokat ugyanugy kell megadnunk, mint bármely más PL/I program fut-

tatásakor.

Ha a SAVE utasitást használjuk /háttértárat használunk/, meg kell adnunk a SYSUT1 DD utasitást is. A SYSUT1 adatállományt direkt elérésü periférián /lemezen/ kell kijelölni, DCB=(BLKSIZE=829,RECFM=F).

4.5 Katalogizált eljárás egy FORMAC job futtatásához

A KFKI Számitóközpontjának R40 számitógépén a következő katalogizált JCL eljárással futtathatunk FORMAC jobokat /az eljárás a SOFT.PROCLIB eljáráskönyvtárban található/:

```
//FORMAC PROC
//FORM EXEC PGM=MINIMAC,REGION=160K
//STEPLIB DD DSN=SOFT.FORMAC.LOADLIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT3 DD UNIT=SYSSQ,DCB=(RECFM=FB,LRECL=88,BLKSIZE=88),
//   SPACE=(88,(4000,2000)),DSN=&&SRCE,DISP=(NEW,PASS)
//PL1L EXEC PGM=IEMAA,COND=(16,EQ,FORM),REGION=160K,
//   PARM='SOURCE,SORMGIN=(1,80)'
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//   SPACE=(80,(250,100))
//SYSUT3 DD UNIT=SYSSQ,SPACE=(80,(250,250)),SEP=SYSPRINT
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(60,60),,CONTIG),
//   SEP=(SYSUT3,SYSPRINT,SYSLIN)
//SYSIN DD DSN=&&SRCE,DISP=(OLD,DELETE)
//LKED EXEC PGM=IEWL,PARM='LIST',
//   COND=((9,LT,PL1L),(16,EQ,FORM))
//SYSLIB DD DISP=SHR,DSN=SOFT.FORMAC.LOADLIB
//   DD DSN=SYS1.PL1LIB,DISP=SHR
//SYSLMOD DD DSN=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//   SPACE=(3520,(50,20,1),RLSE)
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),
//   SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//   DD DDNAME=SYSIN
```

```
//GO EXEC PGM=*.LKED.SYSLMOD,REGION=200K,
//   COND=((9,LT,LKED),(9,LT,PL1L),(16,EQ,FORM))
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,DCB=(RECFM=F,BLKSIZE=829),
//   SPACE=(829,(1000))
```

Megjegyzendő, hogy a STEPLIB kártya is a procedurában szerepel. A DSNAME=SOFT.FORMAC.LOADLIB load modul könyvtár egy rezidens rendszerlemezen lévő, katalogizált adatállomány.

4.6 FORMAC job-ok összeállitása

A közönséges FORMAC jobok forditásához, szerkesztéséhez és végrehajtásához az alábbi kártyacsomagot kell összeállitani:

```
//jobnév  JOB ...
//   EXEC FORMAC
//FORM.SYSIN DD *
     PROG: PROCEDURE OPTIONS(MAIN);
           FORMAC_OPTIONS;  /kötelező!/
           OPTSET (...);    /opcionális/
           CONVERT ...;     /opcionális/
            :
           END PROGRAM;
/*
//LKED.ddnév DD DSNAME=filenév,DISP=OLD
//LKED.SYSIN DD *
     INCLUDE ddnév(modulnév)
/*
//GO.SYSIN DD *
     bemenő adatok
/*
//
```

} ha semmilyen speciális szerkesztési utasitást nem használunk, ezek az utasitások kihagyhatók.

} ha nincs bemenő adat, akkor ezek az utasitások elhagyhatók

A FORMAC utasitások - ugyanugy, mint a PL/I utasitások alapértelmezés szerint - a 2-72. karakterpoziciókat használhatják. A forrásprogramot lezáró

```
/*   /End-Of-File/
```

karaktereknek az 1-2. karakterpoziciókra kell kerülniük.

5. Mintaprogramok

A következőkben két teljes FORMAC programot mutatunk be.

5.1 Legendre-polinomok generálása

A következő FORMAC program az első tiz Legendre-polinomot generálja az alábbi két módszerrel:

1. Differenciálással /generátorfüggvénnyel/:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n(x^2-1)^n}{dx^n}$$

2. Rekurziv összefüggéssel:

$$Q_n(x) = \frac{2n-1}{n} Q_{n-1}(x) - \frac{n-1}{n} Q_{n-2}(x), \quad /n \geq 2/$$

$$Q_o(x) = 1,$$

$$Q_1(x) = x.$$

A program ellenőrzi a $P_n(x)=Q_n(x)$ egyenlőség teljesülését és kinyomtatja az eredményt.

```
LEGR: PROCEDURE OPTIONS(MAIN);
FORMAC_OPTIONS;
OPTSET (LINELENGTH=72);
OPTSET (EXPND);
   PUT PAGE;
   /*LEGENDRE POLINOMOK GENERÁLÁSA AZ ELSŐ MÓDSZERREL*/
   DO N=O TO 1O;
   LET(P("N")=DERIV((X**2-1)**"N",X,"N")/(2**"N"*FAC("N")));
   END;
   /*LEGENDRE POLINOMOK GENERÁLÁSA A MÁSODIK MÓDSZERREL*/
   LET(Q(O)=1; Q(1)=X);
   DO N=2 TO 1O;
   LET(N="N";
   Q(N)=(2*N-1)/N*X*Q(N-1)-(N-1)/N*Q(N-2));
   END;
   /*P(N)=Q(N) ELLENŐRZÉSE ÉS AZ EREDMÉNY KINYOMTATÁSA*/
```

```
      PUT LIST('LEGENDRE POLYNOMIALS'); PUT SKIP(3);
      DO N=0 TO 10;
      LET (N="N");
      IF IDENT (P(N);Q(N)) THEN PRINT_OUT(P(N));
           ELSE STOP;
      END;
      END LEGR;
/*
```

Megjegyzés:

A JCL utasitások és a file végét jelző /* utasitás a kártya első oszlopában kell, hogy kezdődjenek; a FORMAC program utasitásai csak a 2-72. oszlopokat használhatják.

A LEGR program eredménye a SYSPRINT periférián:

```
LEGENDRE POLYNOMIALS
P(0) = 1
P(1) = X
P(2) = 3/2X**2-1/2
P(3) = -3/2X+5/2X**3
P(4) = -15/4X**2+35/8X**4+3/8
P(5) = 15/8X-35/4X**3+63/8X**5
P(6) = 105/16X**2-315/16X**4+231/16X**6-5/16
P(7) = -35/16X+315/16X**3-693/16X**5+429/16X**7
P(8) = -315/32X**2+3465/64X**4-3003/32X**6+6435/128X**8+35/128
P(9) = 315/128X-1155/32X**3+9009/64X**5-6435/32X**7+12155/128X**9
P(10) = 3465/256X**2-15015/128X**4+45045/128X**6-109395/256X**8+
        46189/256X**10-63/256
```

5.2 Közönséges differenciálegyenlet megoldása

A következő példa a FORMAC73 rendszerben bevezetett FORMAC eljárások használatát szemlélteti [8]. Egyszersmind néhány programozási fogást is bemutatunk.

A feladat az

$$y' = y^2 + x^2$$

differenciálegyenlet megoldása az

$$y^{(o)} = c$$

kezdeti feltétel mellett. A megoldást az együtthatók összehasonlításának módszerével keressük. Az y függvényt az

$$y = \sum_{i=o}^{lo} a_i x^i$$

alakban állitjuk elő. Ezt differenciálva,

$$y' = \sum_{i=o}^{lo} i a_i x^{i-1}$$

adódik. /A kezdeti feltétel szerint $a_o=c$./

Ezután az

$$-y' + x^2 + y^2 = 0$$

egyenlőségben elvégezve a koefficiensek összehasonlitását kiszámitjuk az a_i együtthatókat.

A FORMAC program listája:

```
 1  MAIN: PROC OPTIONS(MAIN);
 2  FORMAC_OPTIONS;
 3    DCL(I,N) BIN FIXED(31);
 4    F_FUNCTION(COEFVEC);
 5    OPTSET(LINELENGTH=63);
 6    LET(FNC(EL)=ARG($(1)+2,$(2)));
 7    N=10;
 8    LET(Y=C);
 9    DO I=1 TO N;
10      LET(Y=Y+A("I")*X**"I");
11    END;
12    PRINT_OUT(Y); PUT SKIP(2);
13    OPTSET(EXPND);
14    LET (Z=COEFVEC(-DERIV(Y,X)+X**2+Y**2,X,"N"));
15    DO I=0 TO N-1;
16      LET(I="I");
17      LET(T=COEFVEC(EL(I,Z),A(I+1)));
18      LET(AA=A(I+1));
19      LET(A(I+1)=-EL(0,T)/EL(1,T));
20      LET(Z=EVAL(Z,AA,A(I+1)));
21      PRINT_OUT(A(I+1));
```

```
22        ATOMIZE(A(I+1));
23      END;
24      COEFVEC: F_PROC(E,X,H)LOCAL(L);
25        DCL(I,L,H) BIN FIXED(31);
26        IF IDENT(H;DENARGS) THEN LET (H=HIGHPOW(E,X));
27        LET(L=LOWPOW(E,X));
28        H=INTEGER(H);
29        L=INTEGER(L);
30        LET(DENARGS=COEFF(E,X**H));
31      DO I=H-1 TO L BY -1;
32        LET(DENARGS=CHAIN(COEFF(E,X**"I"),DENARGS));
33      END;
34      LET(DENARGS=VEC.(EVAL(E,X,O),DENARGS));
35      F_RETURN;
F_RETURN NOT FOLLOWED BY AN EXPRESSION
36      F_END COEFVEC;
37      END MAIN;
```

Az utasitás sorszámok nem tartoznak a forrásprogramhoz; ezeket a FORMAC preprocesszor nyomtatja ki az egyes sorok elején. Ugyancsak a preprocesszor nyomtatja ki az F_RETURN NOT FOLLOWED... figyelmeztető üzenetet is.

A programban a COEFVEC nevü FORMAC eljárás egy kifejezésben szereplő változó hatványainak együtthatóit állitja elő. Ezeket a CHAIN müvelet segitségével egy listára füzi fel és a VEC függvényváltozó paraméterlistájára teszi. Ezután a DENARGS rendszerváltozó kapja meg az igy előállitott értéket. Vegyük észre, hogy az eljárás deklarációjában az F_RETURN utasitásban nem szerepel sem változó, sem kifejezés. Ezt a listán egy figyelmeztető üzenet is jelzi. Az eljárás tehát a 2.5 pont szerint a DENARGS változó értékét fogja függvényértékként a hivó programnak visszaadni.

A főprogramban definiáltuk az EL(i,z) függvényt: ez a z kifejezés (i+2)-dik argumentumát választja ki. z-nek természetesen függvénykifejezésnek kell lennie.

A programban az EL függvényt minden esetben a COEFVEC függvény eredményére alkalmazzuk, tehát a VEC függvényváltozó argumentumlistájáról választunk ki vele egy-egy elemet. Vegyük észre, hogy VEC-nek nincs is semmi más szerepe a programban ezen kivül; a COEFVEC felfüzi VEC argumentumlistájára az elemeket, EL pedig le-

emeli őket. Ezért maradhat VEC függvényváltozó, mert csak formális szerepe van.

Még egy programozási fogásra hivjuk fel a figyelmet. A COEFVEC FORMAC eljárásnak deklarációja szerint három paramétere van. A 17. sorban azonban csak két aktuális paraméterrel hivjuk. Ilyenkor a FORMAC rendszer ismét a DENARGS változó értékét veszi a hiányzó paraméter értékeként. Az eljárás erre fel is van készülve: egy olyan utasitással kezdődik, amely vizsgálja, hogy a harmadik paraméter egyenlő-e DENARGS-sal. Példánk tehát a változó hosszuságu paraméterlista használatát is bemutatja.

A program eredménylistája:

$$Y=C+A(5)X^5+A(6)X^6+A(7)X^7+A(8)X^8+A(9)X^9+A(10)X^{10}+A(1)X +$$
$$+X^2A(2)+X^3A(3)+X^4A(4)$$

$$A(1) = C^2$$

$$A(2) = C^3$$

$$A(3) = C^4+1/3$$

$$A(4) = 1/6\ C+C^5$$

$$A(5) = 1/5\ C^2+C^6$$

$$A(6) = 7/30\ C^3+C^7$$

$$A(7) = 4/15\ C^4+C^8+1/63$$

$$A(8) = 1/56\ C+3/10\ C^5+C^9$$

$$A(9) = 8/315\ C^2+1/3\ C^6+C^{10}$$

$$A(10)= 143/4200\ C^3+11/30\ C^7+C^{11}$$

6. Összefoglalás

A FORMAC nyelvnek az algebrai manipuláció, a szimbolikus számitások területén uttörő szerepe volt. Sok fontos alapelvet, melyeket a később kidolgozott rendszerek is felhasználtak, a FORMAC szerzői dolgoztak ki először. A modernebb nyelvek konkurrenciája a FORMAC nyelvet később némileg háttérbe szoritotta. Az 1973-ban végrehajtott átdolgozás azonban ujabb lendületet adott a FORMAC alkalmazásainak. Ennek a nyelvnek ma is vannak olyan jó tulajdonságai, amelyek használatát indokolják. Ezek:

- mivel egy magasszintü nyelvre - a PL/I-re - épül, a programozónak a PL/I valamennyi lehetősége /karakterkezelés, numerikus számitások/ a rendelkezésére áll. Ez igen fontos, mivel sok probléma megoldása kivánja a szimbolikus és numerikus módszerek együttes alkalmazását [17]. Más formulamanipulációs nyelvek alkalmazásánál gondot is okoz a numerikus számitások lassusága, korlátozott lehetősége;
- memória- és futási időigénye viszonylag kicsiny. Ez annak köszönhető, hogy a FORMAC nem értelmező programmal müködik: továbbá, a változó méretü formulák kezeléséhez szükséges dinamikus memóriakezelést nem egy általános listakezelő rendszer, hanem specializált szubrutinok végzik. Igy, mig egy REDUCE2 program minimális memóriaigénye IBM 360/370 vagy ESzR gépen 300 KByte, addig egy nem tul hosszu FORMAC program memóriaigénye 100-200 KByte között van. Hasonló arányok érvényesek a futási időkre is /természetesen a feladattól függően/.

A FORMAC nyelv tehát hatékony, jól használható eszköz lehet a szimbolikus számitásokat végző fizikus, mérnök, matematikus kezében.

Befejezésül köszönetet mondunk Dr. H.W. Homrighausen-nak /Kernforschungsanlage Jülich, NSzK/, aki a FORMAC73 rendszert rendelkezésünkre bocsátotta.

I. Függelék
A FORMAC nyelvben érvényes korlátozások

A FORMAC nyelvben különböző korlátozások érvényesek, amelyekre a programozónak a programok irásánál tekintettel kell lennie. Ha ezeket a korlátozásokat a programozó nem veszi figyelembe, akkor vagy az előfeldolgozás során a preprocesszor ad hibajelzést, vagy a leforditott program hibásan müködik, és futás közben ad hibajelzést.

A FORMAC nyelv korlátozásainak nagy részét ismertettük a megfelelő fejezetekben. Az áttekinthetőség kedvéért itt összefoglaljuk a legfontosabbakat.

Egész számok

Egész tipusu FORMAC konstans vagy egész értékü FORMAC változó értéke legfeljebb 2295 decimális jegyből állhat.

Indexes FORMAC változó indexének abszolut értéke legfeljebb 31622 lehet.

PL/I változó vagy konstans értéke, amelyet FORMAC egész számmá konvertálunk a FIXEDA utasitással, legfeljebb 31622 lehet. Például az

```
I = 100 000;
LET (K=FIXEDA(I));
```

utasitások hibát eredményeznek, mert I értéke nagyobb, mint 31622.

Lebegőpontos számok

Lebegőpontos FORMAC konstans legfeljebb 9 értékes jegyből állhat.

Lebegőpontos FORMAC konstans kitevőjének abszolut értéke nem lehet nagyobb, mint 78.

FORMAC azonositók

FORMAC azonositó /változó vagy eljárás neve/ legfeljebb 8 karakterből állhat.

A név nem tartalmazhatja a (PL/I nyelvben egyébként megengedett) "$" /dollár/, "_" /aláhuzás/ és "@" /egységárjel/ karaktereket. A "$" /dollár/ karakternek a FORMAC nyelvben speciális jelentése van a formális argumentumok jelölésére.

CONVERT utasitás argumentumainak száma

A CONVERT utasitásban a FLOAT listán legfeljebb 99 változónév szerepelhet.
A CONVERT utasitásban a FIXED listán legfeljebb 99 változónév szerepelhet.

Megjegyzések /kommentárok/ és literális karaktersorozatok száma

A megjegyzések /kommentárok/ és a literális karaktersorozatok együttes száma nem lehet több, mint 400.

Háttértárba irható kifejezés mérete

A SAVE utasitással a háttértárba irható kifejezés mérete nem lehet nagyobb, mint 16 384 rekord. /Egy rekord mérete a SYSUT1 DD nevü háttér adatállományban 829 byte./

Forrásprogram mérete

A FORMAC forrásprogram mérete nem lehet nagyobb, mint 32760 karakter. A méretbe csak a szignifikáns karakterek számitanak bele. Ez a korlátozás kb. 1500-2000 forrásnyelvi utasitásból álló programot enged meg.

EDIT opció

Az EDIT opció, azaz

```
OPTSET(EDIT);
```

érvényessége mellett egy kifejezésben a kitevők nyomtatásához szükséges soremelések száma nem lehet több, mint 5. Ha az EDIT opció érvényes, a kifejezésekben a kitevők soremeléssel nyomtatódnak. Például a

```
PRINT_OUT(Y=A**A**A);
```

utasitás eredménye az

$$Y = A^{A^A}$$

formában jelenik meg. A

```
PRINT_OUT(Y=B**B**B**B**B**B**B);
```

utasitás eredményét azonban a fenti korlátozás miatt a FORMAC rendszer nem tudja kinyomtatni: a kitevők kiirásához szükséges soremelések száma ebben az esetben 7 lenne, ami több, mint a megengedett maximális érték. A FORMAC rendszer nyomtatás helyett hibajelzést ad.

Megjegyzés: természetesen az

```
OPTSET (NOEDIT);
```

utasitás érvényessége mellett a fenti utasitást is ki lehet nyomtatni.

II. Függelék
A FORMAC hibajelzései

A/ A preprocesszor által jelzett hibák

A preprocesszc , mint azt a 4.1 pontban mondottuk, nem végez teljes szintaktikus és szemantikus elemzést. Ezért viszonylag kevés hiba derül ki az előfeldolgozás során. A preprocesszornak kevés hibajelzése van, és ezek nagyrészt olyan eseteket jeleznek, amikor a forrásprogram a preprocesszor valamelyik korlátozását tullépi, és a rendszer valamelyik táblázata betelik.

A forrásprogramban elkövetett hibák egy része csak a PL/I fordításkor derül ki. Példaként említünk néhány gyakori hibát:

a/ Ha a programból kifelejtettük a /kötelező/

FORMAC_OPTIONS;

utasitást, a PL/I fordító minden /!/ FORMAC utasitásból generált PL/I utasitást hibásnak jelez. Az ok: a szükséges deklarációkat a FORMAC_OPTIONS hatására épiti be a preprocesszor a leforditott PL/I programba, mivel ez hiányzik, a generált PL/I utasitások deklarálatlan eljárások hivását tartalmazzák.

b/ Ha egy LET utasitás bezáró zárójele hiányzik, a preprocesszor a program hátralévő részét a LET utasitáshoz tartozónak tekinti. A PL/I fordító fogja jelezni, hogy a programot lezáró END; utasitás hiányzik.

Mivel a preprocesszor nem végez teljes szintaktikus elemzést, ezért abból, hogy a FORMAC előfeldolgozás és a PL/I fordítás sikeresen megtörtént, nem következik, hogy a program szintaktikusan helyes. A szintaktikus hibák egy részét csak a program végrehajtása közben észlelik és jelzik a FORMAC rendszer rutinjai. Ezeket a hibajelzéseket a jelen Függelék B. pontja tartalmazza.

A következőkben felsoroljuk a preprocesszor hibajelzéseit. Ha a hibajelzések bármelyike előfordul, az előfeldolgozás eredménytelen lesz /nem keletkezik lefordítható és futtatható PL/I program/.

A hibajelzések egy részénél a preprocesszor kiirja azt a

FORMAC utasitást is, ahol a hibát észlelte. Sok esetben előfordul azonban, hogy az észlelt hiba egy korábbi hiba következménye, igy a "valódi" hibát a program előző részében kell keresnünk.

Fatális hibák

Ezeknek a hibáknak a hatására az előfeldolgozás megszakad.

1. PROGRAM IS TOO LONG. TOTAL NUMBER OF SIGNIFICANT CHARACTERS MAY NOT EXCEED 32760 /APPROXIMATELY 2000 CARDS/
   A program tul hosszu. Több, mint a maximálisan megengedett 32760 karakter /kb. 1500-2000 kártya/. A programot röviditeni kell, vagy szegmensekre kell bontani.
2. ERROR_TOO MANY FIXED VARIABLES DECLARED, ONLY 99 ARE ALLOWED
   A CONVERT utasitásban FIXED listán szereplő változók száma legfeljebb 99 lehet.
3. ERROR_TOO MANY FLOAT VARIABLES DECLARED, ONLY 99 ARE ALLOWED
   A CONVERT utasitásban FLOAT listán szereplő változók száma legfeljebb 99 lehet.
4. ERROR_UNMATCHED COMMENT DELIMITER OR QUOTES IN THIS PROGRAM LOOK AT DATA /utasitás/
   IN THE INPUT STREAM
   A COMMENT utasitás elhatároló jelének (/*, */) vagy egy idézőjelnek hiányzik a párja! Teendő: A hibát ki kell javitani, és a programot ujra futtatni.
5. ERROR_MORE THAN 400 COMMENTS OR STRING LITERALS IN PROGRAM, INPUT DATA STARTING AT /utasitás/
   A program több, mint 400 kommentárt vagy literális karaktersorozatot tartalmaz. Teendő: a kommentárok egy részét ki kell hagyni, vagy szegmentálni kell a programot.

Nem fatális hibák

A következő hibajelzések többször is /legfeljebb 15-ször/ előfordulhatnak. 15-nél több előfordulás hatására az előfeldolgozás megszakad. Lefordítható és futtatható program azonban már egy hibajelzés után sem keletkezik.

6. UNMATCHED PARENTHESIS IN THIS INPUT DATA AT OR AROUND /utasitás/
   A FORMAC forrásprogramban egy zárójelnek nincs párja. A hibát ki kell javitani és a programot ujra futtatni.

7. ERROR, IDENTIFIERS OR OPERATORS IN WRONG ORDER IN /utasitás/
   A FORMAC utasitásban azonositók vagy müveleti jelek hibás sorrendben szerepelnek.
8. ERROR_UNMATCHED " IN STATEMENT /utasitás/
   A FORMAC utasitásban egy " jel párja hiányzik.
9. ERROR IN (FIXEDA, IDENT, FLOATA) STATEMENT
   EXPANSION NOT DONE
   Hiba a FIXEDA, IDENT vagy FLOATA utasitások valamelyikében. A FORMAC utasitásokat ki kell javitani és a programot ujra futtatni.
10. ERROR IN OPTSET ILLEGAL OPTION /opció/ ASKED FOR
    Meg nem engedett kulcsszó szerepel az OPTSET utasitásban. Az OPTSET utasitást ki kell javitani és a programot ujra futtatni.

Figyelmeztető üzenetek

Ezek az üzenetek olyan hibákra figyelmeztetnek, amelyeket a preprocesszor javitani tud, vagy olyan esetekre, amikor valamilyen információt nem adtunk meg, és az alapértelmezés lépett érvénybe. Az üzenet kiirása után az előfeldolgozás folytatódik, és eredményeképpen előáll egy PL/I program. Mindig gondosan ellenőrizzük azonban, hogy valóban helyesen szerepel-e az üzenettel jelzett utasitás.

Példák figyelmeztető üzenetekre:

1. INVALID DELIMITER 'x' ASSUMED

A preprocesszor hibás elhatárolójelet talált, és azt az 'x' karakterrel helyettesiti. Például:

```
CONVERT FIXED (I,K);
```

után az üzenet

```
INVALID DELIMITER ';' ASSUMED
```

A CONVERT utasitásban felsorolt változóneveket ugyanis a pontosvessző /;/ karakterrel kell elválasztani. A preprocesszor az utasitást ugy dolgozza fel, mintha az a helyes

```
CONVERT FIXED (I;K);
```

alakban szerepelt volna.

2. F_RETURN NOT FOLLOWED BY AN EXPRESSION

FORMAC eljárásban /F_PROCEDURE/ az F_RETURN kulcsszó után nem szerepel változónév. A függvény a DENARGS rendszerváltozó értékét veszi fel.

B/ Futás közben jelzett hibák

Egy tipikus futás közbeni hibajelzés megjelenési formája:

```
DENO58I    DENAUT    LOG(O)
Y(7) = LOG(X(I))
IHE5OlI CONDITION DENSEM IN STATEMENT OO195
        AT OFFSET + OO48A FROM ENTRY POINT
        DENFMCl
IHE5OlI CONDITION DENERRR IN STATEMENT OO195
        AT OFFSET + OO49O FROM ENTRY POINT
        DENFMCl
```

A hibajelzés egyes részeinek magyarázata a következő:

DENO58I A hiba száma. A felhasználónak meg kell keresnie a hiba számát a táblázatban, ahol megtalálja a hiba részletesebb magyarázatát.

DENAUT Annak a rutinnak a neve, amelyben a hiba előfordult.

LOG(O) A hiba rövid leirása; jelen esetben a LOG függvény argumentuma nulla.

Y(7)=LOG(X(I)) A hibás kifejezés kiirása. Ha a hiba a kifejezés kiértékelése során keletkezett, csak a kifejezés helyesen kiértékelt részét irja ki. Ez lehetővé teszi a felhasználónak, hogy a hibát az utasitásban lokalizálja.

IHE5OlI CONDITION DENSEM IN STATEMENT OO195 ...

A DENSEM feltétel bekövetkezett a jelzett sorban /szemantikus hiba, l. a 3. fejezetet/.

A 3. fejezetben ismertettük a FORMAC rendszer ON feltételeit. Ezek a következők:

- DENSYN - szintaktikus hiba
- DENSEM - szemantikus hiba
- DENSIZE - memória kimerült
- DENSYSM - rendszerhiba
- DENERRR - hiba /az előző feltételek bármelyike bekövetkezett/.

A programozó a PL/I nyelv ON utasitását használhatja arra, hogy a fenti feltételek bármelyikének bekövetkezésekor egy hibafeldolgozó programrészre adhassa át a vezérlést, például:

```
ON CONDITION (DENSYN) GOTO HIBA;
```

ahol HIBA egy PL/I utasitáscimke.

Ha a program végrehajtása közben egy hiba fordult elő, amelynek hatására bekövetkezik a DENxxx ON feltétel, és a programban nem szerepel erre a feltételre vonatkozó ON utasitás, akkor a FORMAC rendszer kiirja a megfelelő hibaüzenetet. Ezt az IHE051I számu PL/I hibaüzenet követi, amelyik kiirja, hogy melyik ON feltétel következett be. Rendszerint két ilyen PL/I hibaüzenetet kapunk, a második a DENERRR feltételre vonatkozik.

A hibajelzések táblázatát a következő formában adjuk meg:

Hiba száma rutin neve - FORMAC ON feltétel
Hibajelzés szövege
A hiba rövid magyarázata, egy egyszerü példa, a felhasználó teendői.

DENOOOI ********-DENSYSM
***** FORMAC SYSTEM ERROR *****
Rendszer hiba. A hibajelzés oka tehát vagy a FORMAC rendszer hibája, vagy egy olyan bonyolult hiba a felhasználó programjában, amelyet a FORMAC rutinok nem tudtak pontosan kielemezni. A forrásprogram figyelmes átolvasásával és egyes "gyanus" utasitások átirásával esetleg ki tudjuk a hibát küszöbölni.

DENOO1I DENINS - DENSYN
NAME IN WRONG ORDER
például: LET(5AB = Y);
Itt 5AB egy meg nem engedett azonositó /számjeggyel kezdődik/.

DENOO2I DENINS - DENSYN
INVALID CHARACTER
például: LET(Y = B*C_A);
Illegális karakter szerepel egy változó nevében /FORMAC változónév nem tartalmazhatja az "aláhuzás" és az egységárjel /"@"/ karaktereket, a "$" jelnek pedig speciális szerepe van a formális argumentumok jelölésére/.

DENOO3I DENINS - DENSYN
NAME TOO LONG
Például: LET(Y = ABCDEFGHIJ+B);
A változó neve tul hosszu, meghaladja a maximális 8 karaktert.

DENOO4I DENINS-DENSYN
. OR CONSTANT IN WRONG ORDER
például: LET(Y = 3E-1OO);
Lebegőpontos konstans kitevője legfeljebb kétjegyü szám lehet.

DENOO5I DENINS-DENSYN
(IN WRONG ORDER
például: LET(Y = 34(A+B));
34 és (között hiányzik egy müveleti jel.

DENOO6I DENINS-DENSYN
CONSTANT TOO LONG
például: LET(A = 1O9876.54321);
Lebegőpontos konstans több, mint 9 számjegyből áll.

DENOO7I DENINS-DENSYN
2 DECIMAL POINTS IN CONSTANT
például: LET (Y=123.456.789);
Egy konstansban két tizedespont szerepel.

DENOO8I DENINS-DENSYN
2 = SIGNS IN EXPRESSION
például: LET(Y=A=B+C);
Egy kifejezésben két egyenlőségjel szerepel.

DENOO9I DENINS-DENSYN
= SIGN IN WRONG ORDER
például: LET(=A+B+C);
Egyenlőség baloldala hiányzik.

DENO1OI DENINS-DENSYN
OPERATOR IN WRONG ORDER
például: LET(Y=A+*A+B);
Két müveleti jel szerepel egymás mellett.

DENO11I DENINS-DENSYN
OPERATOR IN WRONG ORDER ON PDL (USER)
például: LET(Y+X=A+B+C);
Müveleti jel meg nem engedett helyen áll, példánkban egy egyenlőség baloldalán kifejezés szerepel /PDL a Push--Down-List nevének röviditése/.

DENO13I DENINS-DENSEM
CHAIN IN ILLEGAL CONTEXT
például: LET(A=CHAIN(B,C,2);Y=A);

Mivel az A FORMAC változó egy listát tartalmaz, a második utasitás illegális.
Helyesen: Y=(A);

DENO14I DENINS-DENSYN
WRONG NUMBER OF OPERANDS
például: LET(Y=ATAN(A,B,C));
Függvénykifejezésben az előirtnál több vagy kevesebb argumentum szerepel.

DENO15I DENINS-DENSEM
FUNCTION VARIABLE WITH NON-ATOMIC NAME
például: LET(Z=A+B; Y=Z.(X));
A Z-t függvényváltozó neveként használtuk, előzőleg azonban Z-nek már értéket is adtunk, tehát Z nem atom.

DENO16I DENINS-DENSEM
NON-NUMERIC SUBSCRIPT
például: LET(Y=A(B));
Tömbelemnél numerikus index helyett olyan változó szerepel, amelynek értéke nem szám.

DENO17I DENINS-DENSYN
TOO MAY SUBSCRIPTS
például: LET(Y=A(1,2,3,4,5));
Tömbelemnek tul sok indexe van /legfeljebb négy lehet/.

DENO20I DENINS-DENSEM
FAULTY ASSIGNMENT
például: LET(FAC(1)=X);
Meg nem engedett értékadás.

DENO21I DENINS-DENSYN
WRONG NUMBER OF ASSIGNMENTS TO PSEUDOVAR
például: LET(DIFF(X,Y)=...);
Pszeudováltozónak nem megfelelő számu argumentuma van. DIFF-nek csak egy argumentuma lehet, egy függvényváltozónév.

DENO22I DENINS-DENSYN
TRACEBACK
A DENINS szubrutin által hivott valamelyik rutinban fordult elő a hiba. Ezzel együtt egy másik hibajelzésnek is következnie kell, amely további magyarázatot ad.

DENO23I DENINS-DENSEM
NON-ATOMIC ARGUMENT OF PSEUDO-VARIABLE
például: LET(Z=A+B; DIFF(Z)=D+E+F);
Pszeudováltozó argumentuma nem atomi.

DENO24I DENINS-DENSEM
ATTEMPTED ASSIGNMENT TO RESERVED NAME
például: LET(EXP=A+B);
Értékadó utasitás baloldalán fenntartott név szerepel /EXP fenntartott név/.

DENO25I DENINS-DENSEM
NAME IN INAPPROPRIATE PLACE
például: LET(Y=EVAL+SIN);
Név nem megfelelő helyen szerepel /példánkban EVAL és SIN argumentum nélkül/.

DENO27I DENINS-DENSYN
NO = SIGN AND NOT A NAME
például: LET(FAC(A));
Értékadó utasitásból az értékadás jele /=/ és a baloldalon álló változó neve hiányzik.

DENO28I DENINS-DENSYN
UNBALANCED PARENTHESES
például: LET(Y=D*(A+SIN(B));
Egy zárójelnek nincs párja.

DENO29I DENINS-DENSYN
PSEUDO-VARIABLE AFTER = SIGN
például: LET(Y=DIFF(F.(X)));
Pszeudováltozó csak baloldalon szerepelhet.

DENO3OI DENINS-DENSIZE
PUSHING OVER THE TOP OF A PDL
A FORMAC rendszer PDL táblázata /Push-Down-List, amely egy fix méretü tábla/ betelt. A felhasználó az összetett kifejezést, amellyel kapcsolatban a hibajelzést kapta, bontsa több egyszerübb kifejezésre.

DENO32I DENEVL-DENSYN
ODD NUMBER OF ARGUMENTS
például: LET(Y=EVAL(A*X**2,A,3,X));
EVAL-ban nem megfelelő számu argumentum szerepel.

DENO35I DENOUT-DENSEM
NOT LET VARIABLE
például: PRINT_OUT(SIN);
PRINT_OUT utasitásban nem FORMAC változó vagy kifejezés szerepel.

DENO37I DENOUT-DENSIZE
PAST END OF STRING
például: DECLARE V CHAR(20);
CHAREX(V=Y);
ahol az Y kifejezés több mint 20 karakterből áll. A felhasználó egy kifejezést karaktersorozattá próbál konvertálni, és az eredmény karaktersorozat hosszabb, mint azon karakter tipusu változó, amelybe az eredményt tenni akarja. Javitani ugy lehet ezt a hibát, hogy a karakter tipusu változót nagyobb méretre deklaráljuk.

DENO38I DENOUT-DENSYN
TRACEBACK
Hiba a DENOUT szubrutin által hivott valamelyik szubrutinban. Ezzel együtt egy másik hibajelzésnek is következnie kell, amely további magyarázatot ad.

DENO41I DENOUT-DENSIZE
EXPRESSION TOO HIGH FOR EDITING
például: Ha az EDIT opció érvényes, és a következő utasitást adjuk ki:
PRINT_OUT(Y=A**A**A**A**A**A**A**A);
Minden kitevőnek eggyel magasabb sorba kellene nyomtatódnia, a soremelések lehetséges maximális száma pedig 5. A fenti kifejezés tehát csak OPTSET(NOEDIT); esetén nyomtatható.

DENO42I DENAPA-DENSIZE
NUMBER TOO LARGE FOR CONVERSION
például: LET(Y=A(31623));
Az indexként megadott egész szám tul nagy, a legnagyobb megengedett index 31622.

DENO43I DENAPC-DENSEM
NEGATIVE ARGUMENT
például: LET(Y=FAC(-3); vagy
LET(Y=COMB(-3,2));

A FAC vagy COMB rutinban negativ argumentum szerepel.

DENO44I DENAPC-DENSIZE
CONSTANT OVERFLOW
például: LET(Y=FAC(920));
Az egész tipusu eredmény tul hosszu /FAC(920) több, mint 2295 jegyü szám/.

DENO47I DENDRV-DEMSEM
VARIABLE OF DIFFERENTIATION NOT A VARIABLE
például: LET(Y=A*X; Z=DERIV(SIN(X),Y,2));
A DERIV rutinban a változó, amely szerint differenciálunk, csak atomi lehet. A hibát kiküszöbölhetjük, ha az első utasitást az

Y=X;

utasitással cseréljük ki.

DENO48I DENDRV-DENSEM
NEGATIVE ORDER OF DIFFERENTIATION
például: LET(Z=DERIV(SIN(X),X,-2));
A DERIV rutinban a differenciálás rendjét definiáló paraméter negativ.

DENO51I DENAFL-DENSEM
ARGUMENT NOT A CONSTANT
például: LET(Y=A);Z=ARITH(Y);
Az ARITH rutin argumentumának értéke nem egy FORMAC konstans.

DENO52I DENAFX-DENSEM
ARGUMENT NOT A CONSTANT
például: LET(Y=A); J=INTEGER(Y);
Az INTEGER rutin argumentumának értéke nem egy FORMAC konstans.

DENO53I DENARG-DENSEM
ARGUMENT NOT A CONSTANT
például: LET(N=A; Y=ARG(N,SIN(X)));
Az ARG rutin első argumentumának értéke nem egy FORMAC konstans.

DENO54I DENARG-DENSEM
NEGATIVE ARGUMENT
például: LET(Y=ARG(-1,SIN(X)));
Az ARG rutinban az első paraméter negativ.

DENO56I DENAUT-DENSEM
DIVISION BY ZERO
például: LET(Y=A/0);
Nullával való osztás.

DENO58I DENAUT-DENSEM
LOG(0)
A hibajelzés a LOG(0), LOG2(0), LOG10(0) függvényértékek generálásakor fordul elő, függetlenül attól, hogy az OPTSET(TRANS) vagy OPTSET(NOTRANS) opció érvényes-e.

DENO59I DENAUT-DENSEM
LOG(-CONS)
ahol CONS egy konstans szám.
például: LET(A=LOG(-3.14));
Mint az előző hibajelzés, ez is előfordulhat akár az OPTSET(TRANS), akár az OPTSET(NOTRANS) mellett is.

DENO84I DENTC5-DENSIZE
NO MORE FREELIST AVAILABLE
A FORMAC munkaterülete kimerült. Ilyenkor a felhasználónak a szükségtelen vagy inaktiv kifejezéseket a munkaterületről vagy a SAVE, vagy az ATOMIZE rutinokkal törölnie kell. A programot ujra kell futtatni, esetleg megnövelt REGION paraméterrel.

DENO88I DENRPL-DENSYN
ODD NUMBER OF ARGUMENTS
például: LET(Y=REPLACE(A*X**2,A,SIN(X),X));
Nem megfelelő számu argumentum szerepel a REPLACE rutinban.

DENO96I DENCPI-DENSIZE
OUT OF ROOM
Egy kifejezést, amelyet a SAVE utasitással vittünk ki a háttértárba, be akartunk hozni a memóriába, hogy feldolgozzuk, de nincs elég hely ehhez a memóriában. A felhasználónak a SAVE vagy az ATOMIZE utasitással kell helyet felszabaditania a memóriában.

DENO98I DENCPO-DENSIZE
NO SYSUT1
A FORMAC program nem tudja végrehajtani a SAVE utasitást; meg kell adnunk a SYSUT1 DD utasitást. Ellenőrizni kell

a GO lépés DD utasitásait a FORMAC eljárásban.

DEN099I DENDGT-DENSYSM
I/O ERROR ON SYSUT1
Input/output hiba a SAVE utasitás végrehajtása közben.
A futtatást meg kell ismételni.

DEN100I DENDNT-DENSYSM
UNABLE TO OPEN SYSUT1
A FORMAC SAVE utasitás nem tudja megnyitni a SYSUT1 adatállományt. Ellenőrizni kell a GO lépés DD utasitásait a FORMAC eljárásban.

DEN101I DENDNT-DENSYSM
SYNCHRONOUS ERROR ON SYSUT1
Input/output hiba a SAVE utasitás végrehajtása közben.
A programot ujra kell futtatni.

DEN102I DENDPT-DENSIZE
MORE THAN 16384 RECORDS NEEDED
A kifejezés, amelyet a SAVE utasitás segitségével háttértárba akartunk kiirni, tul nagy. Több kisebb kifejezésre kell felbontani, és a programot ujra futtatni.

DEN104I DENDPT-DENSIZE
OUT OF ROOM ON SYSUT1
Az elhelyezni kivánt kifejezés számára nincs elég hely a SYSUT1 adatállományban.

DEN111I DENFXA-DENSIZE
NUMBER TOO LARGE FOR CONVERSION
például: A=788941; LET(Y=FIXEDA(A));
31622-nél nagyobb egész tipusu PL/I változót vagy konstanst nem lehet FORMAC egész számmá konvertálni.

DEN112I DENFLX-DENSEM
ERROR IN ARGUMENTS OF DRV
például: LET(Y=DRV(F.(X),X));
A második paraméterben X helyett $(1)-nek kell állnia, mert a DRV rutin függvényváltozóknak argumentumaik szerinti differenciálására szolgál.

DEN113I DENENT-DENSEM
0**0
A felhasználó a 0**0 kifejezést generálta.

III. Függelék
A FORMAC73 rendszer

Az alábbiakban K. Bahr [8] cikke nyomán összefoglalóan felsoroljuk azokat a főbb változtatásokat, bővitéseket, amelyekben a FORMAC73 rendszer eltér az eredeti PL/I-FORMAC rendszertől. Zárójelben megadjuk a fejezet/ek/ számát, ahol a változást, bővitést ismertettük.

a/ Függvények argumentumainak "késleltetett" kiértékelése. A függvények argumentumait a FORMAC73 rendszer nem a függvény definiálásakor, hanem a függvény hivásakor értékeli ki /1.5.3 pont/.

b/ EVAL és REPLACE szintaxisának módositása. Az EVAL rutinban a $név jelölés bevezetése az argumentumokra a $(n) jelölés helyett, rekurziv hivás lehetősége: a REPLACE rutinban a $név argumentumok használata /1.8.2 pont/.

c/ GCF és GCFOUT függvények, kifejezések legnagyobb közös tényezőjének meghatározására és kiemelésére /1.8.1 pont/.

d/ OPTSET(TRUNC=n); utasitás polinomok, illetve hatványsorok csonkitására /1.9 pont/.

e/ FORMAC eljárások, lokális FORMAC változók /F_FUNCTION, F_PROCEDURE, F_END, F_RETURN, LOCAL/ /2.5 és 5.2 pont/.

Irodalom

[1] Tobey, R., Baker, J., Crews, R., Marks, P., Victor, K.: PL/I-FORMAC Interpreter, 1967.

[2] MacDonald, W.D., Truffyn, N.: FORMAC Programmer's Guide. Sandford Fleming Laboratory, University of Toronto, Toronto, Canada, 1968.

[3] Sammet, J.E., Bond, E.R.: Introduction to FORMAC. IEEE Trans. Electron. Computers. EC-13, No.4. /1964/

[4] Tobey, R.G., Bobrow, R.J., Zilles, S.: Automatic Simplification in FORMAC. Proc. Fall Joint Comp. Conf. Vol. 27. 1965. pp. 37-52. Spartan Books

[5] Tobey, R.G.: Eliminating Monotonous Mathematics with FORMAC. Comm. ACM 9 /1966/

[6] Tobey, R.G.: Experience with FORMAC Algorithm Design. Comm. ACM 9 /1966/

[7] Sammet, J.E.: Formula Manipulation by Computer. Advances in Computers /ed.: F.Alt and Rubinoff/ Vol. 8. pp.47-1o2. 1967. Academic Press

[8] Bahr, K.: Utilizing the FORMAC Novelties. SIGSAM Bulletin, No. 33. Febr. 1975. pp. 21-24.

[9] Sammet, J.E.: Programming Languages. History and Fundamentals. Prentice Hall, 1969.

[10] Sconzo, P., Le Shack, A.R., Tobey, R.G.: Symbolic computation of the f and g series by computer. Astron.Journal 70 /1965/

[11] Rudnicki, Bujnowski, G.: Explicit Formulae for Clebsch--Gordan Cefficients. /A FORMAC program/ Computer Phys. Comm. 10. 245-250 /1975/

[12] Moses, J.: Automatic Simplification: a Guide to the Perplexed. Comm. ACM 14. 527-537 /1971/

[13] Hearn, A.C.: REDUCE2 User's Manual.
UCP-19, University of Utah, Salt Lake City, 1973.

[14] Hall, A.D.: The ALTRAN System for Rational Function Manipulation.
Comm. ACM 14. 517-521 /1971/

[15] PL/I Language Reference Guide, GC28-8201

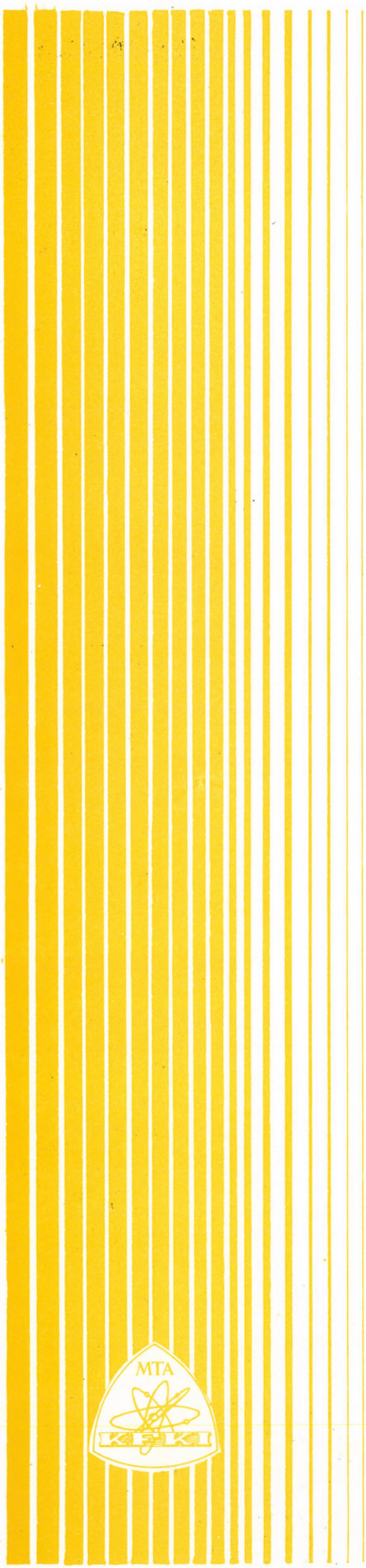
[16] PL/I Programmer's Guide, GC28-6594-8

[17] Brown, W.S., Hearn, A.C.: Application of Symbolic Algebraic Computation.
Computer Phys. Comm. 17. 207-215 /1979/

Tárgymutató

fejezet száma

63.232



Kiadja a Központi Fizikai Kutató Intézet
Felelős kiadó: Lőcs Gyula
Szakmai lektor: Hegedüs Csaba
Példányszám: 405 Törzsszám: 82-234
Készült a KFKI sokszorosító üzemében
Felelős vezető: Nagy Károly
Budapest, 1982. április hó